Understanding GPU Kernels with Instrumentation

Yumeng Liu Advisor: John Mellor-Crummey Rice University

Scalable Tools Workshop 2025



Performance Analysis on a GPU

- Current tools can provide \bullet
 - Traces of GPU functions (kernel launch, memory copies, etc.)
 - High-level metrics of GPU kernels (GPU utilization, bandwidth utilization, etc.) \bullet
 - Detailed metrics per instruction (instruction counts, etc.) ullet
 - Traces of metrics
- A missing feature
 - Traces of GPU device function calls inside kernels over kernel execution time
 - Why? Behavior of complex kernels may differ greatly across threads

Complex GPU-Accelerated Application: Quicksilver

- functions
- High-level structure of the GPU kernel
 - Each GPU thread works on one particle
 - Each particle keeps evolving until a termination condition is met
 - Work for particles differ between threads

Potential load imbalance, optimization opportunity

• A proxy application for a dynamic Monte Carlo particle transport code Mercury, consisting of a single GPU kernel with thousands of lines of code that invokes other GPU device

```
cycle_tracking() {
  for all particles {
    do {
      compute distance to census
      compute distance to facet
      compute distance to reaction
      do segment with shortest distance
      increment tallies
      until census, absorbed, escaped
```





- Trace GPU device function calls within kernels to understand thread behavior
- Explore potential optimizations

Outline

- Our trace tool lacksquare
- Potential optimization opportunities of Quicksilver found with the trace tool •
- Exploration of an optimization opportunity ullet
- Work stealing algorithm among GPU threads lacksquare
- Summary \bullet





Outline

- **Our trace tool**
- Potential optimization opportunities of Quicksilver found with the trace tool
- Exploration of an optimization opportunity
- Work stealing algorithm among GPU threads lacksquare
- Summary \bullet



Instrumentation of GPU Binaries

- Use **NVBit** for dynamic binary instrumentation on NVIDIA GPUs •
- Add instrumentation to trace function invocations on each thread \bullet
 - Instrumentation points ullet
 - Before a call (CALL) lacksquare
 - Upon function entry
 - Before function exit (RET, EXIT) \bullet

Instrumentation of GPU Binaries

- At each instrumentation point, emit a record containing
 - Warp ID + Mask => Thread ID who
 - Timestamp when
 - Address (Special: EXIT will be 0) where



Moving Trace Records to CPU

- Use ChannelDev/ChannelHost objects from NVBit to move records back to CPU
 - One buffer on GPU, one buffer on CPU, flush the buffer when it's full



Processing Trace Stream on CPU

- \bullet
- \bullet



Input: a stream of records containing who (thread id), where (function), when (timestamp)

Output: Calling Context Tree (CCT) and trace, using HPCToolkit measurement format

Callsite information differentiates multiple calls by a function to the same callee



Processing Trace Stream on CPU

- Input: a stream of records containing who (thread id), where (function), when (timestamp)
- Output: Calling Context Tree (CCT) and trace, using HPCToolkit measurement format



- [timestamp, CCT Node ID]
- [t0, 0] [t1, 1] [t2, 3] [t3, 1] [t4, 0] [t5, 2] [t6, 4] ...

Post-mortem Analysis

- Analyze the GPU binary with hpcstruct
- Interpret the trace data with hpcprof
- View the traces in *hpcviewer*



Outline

- Our trace tool ullet
- Potential optimization opportunities of Quicksilver found with the trace tool
- Exploration of an optimization opportunity •
- Work stealing algorithm among GPU threads \bullet
- Summary \bullet





- 100,000 particles
- 100,096 threads
- First impression
 - A lot of idleness
 - Obvious load imbalance





00ms	110ms	120ms	130ms	140ms	150ms	160ms
			•			
	<u>1</u> 1					
		-				
	-					
	-					

- 100,000 particles
- 100,096 threads
- First impression
 - A lot of idleness
 - Obvious load imbalance
 - Roughly half threads start late









- Two imbalance
 - Imbalance between tasks
 - Imbalance between
 cases within each
 iteration





- Two imbalance
 - Imbalance between tasks
 - Imbalance between
 cases within each
 iteration
- Two optimization opportunities
 - Compact tasks among threads





- Two imbalance \bullet
 - Imbalance **between** tasks
 - Imbalance between cases within each iteration
- Two optimization ulletopportunities
 - Compact tasks among threads
 - Each thread executes short cases until the next case is the long one, and all the threads work on the long case together



Outline

- Our trace tool lacksquare
- Potential optimization opportunities of Quicksilver found with the trace tool \bullet
- **Exploration of an optimization opportunity**
- Work stealing algorithm among GPU threads lacksquare
- Summary \bullet



Exploration with Mini Proxy App

- N threads and N tasks, each thread works for one task, N = 100,096 \bullet
- Each task can be either SHORT (5 * 100,000 additions) or LONG (100 * 100,000 additions) \bullet
- Terms: ullet
 - Even threads: threads with even id
 - Odd threads: threads with odd id ullet
 - First half threads: threads with id in [0, N/2)
 - Second half threads: threads with id in [N/2, N)



Mini Proxy App - Potential Benefits

• Same amount of total workload, different threads assignment (who get what task)

Version	LONG task	SHORT task
odd-even	even threads	odd threads
long-short	first half	second half
short-long	second half	first half



Mini Proxy App - Version odd-even



	<pre>time: 336.0 ms time: 234.2 ms time: 233.5 ms time: 233.2 ms time: 233.5 ms Correctness check: PASSED</pre>
200ms 220ms 240ms 260ms 280ms 300ms	



Mini Proxy App - Version long-short

Time Range: [0ms, 150ms]

Cross Hair: (75ms, THREAD 49892)



time: time: time: time: time: time: time: time:	197.5 ms 147.6 ms 120.7 ms 120.6 ms 120.8 ms 120.5 ms 120.5 ms 120.5 ms 120.5 ms tness check:	PASSE



Mini Proxy App - Version short-long



	time: 255.3 ms time: 157.8 ms
	<pre>time: 157.1 ms time: 157.1 ms time: 157.1 ms time: 157.1 ms time: 156.9 ms time: 157.5 ms time: 157.1 ms time: 157.2 ms time: 157.2 ms Correctness check: PASSED</pre>
110ms 120ms 130ms 140ms 150ms 160ms 170ms	



Compact Work is Faster

odd-even

and the second s	5.27 Gh 4 B 7 3 8 3	A 197.617 C 19.1 11	ana wasan sa s										
n der Standstationen Bernig Harfundet auf B													
<u>ja sereta notestas</u>	<u> Markana</u> n	0.0 7 <u>-</u> 1			_								
A some associate and the second	and and	<u></u>											
ferson des consesses													
Contractor and a line of the contraction		an a											
Jan Charles and California California C	Wester Barry	a Collection		ودين فتقرر									
A CARACTER AND A CARACTER	an a		na - Maraka Baratat yan Kabupatén Kabupatén K	n an an trainn an Na Stainneachan	an a	1.111.111.111							
A she have be been shown in	deg a chi e see		مة المراجع بالمرجع الم	a si di si si s	open de Neder	vit thu its							
The and the second second	ar y Valer I Ar	ah kang palah ka	- 11 esti (1917)	1. A. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.	lands anna	tet av ter ter er	and the second						
- The second second Second second	alla an	an each raile	1.121.121	4 / 4 / - 6 / 4	1. I. I. I. I. I.	a barring a start of the		-					
a ng kang pang pang pang pang pang pang pang p						and an easy of the							
。在1997年1月1日,1997年1月1日,1月1日日日 1月1日—————————————————————————————		and the second											
A CAR DEPARTURE AND CARE	e the the set	$p_{\rm eff}^{\rm eff}$ is the set of $p_{\rm eff}^{\rm eff}$	19. mar 19. ma 19. mar 19. mar				-						
- Alfred and a second co	62 S 2 2 3 S 6 C	and a second second	1997 - 1997 -	211.14.253		6 S S S S S	n de de set set	1. Y. A. A. A.	the second second				
			na george i se	$f \in \mathcal{F}$	2008-041 1		let effective a						
			a a de la construcción Nome			an an tha an an		· · · · · ·					
n self felse in Azhandel felse na 1841. 1938 -		ang nga tang kun tana	a se este a de se	e y haide.	i este da la c	da ya sa ƙira	111-11-1	- Artist Artis					
an a' fhaille bha gu dhar ga bhrainn ar An Anna an Anna		n an an tha an tha Tha an tha an t	en gineration.	e i device Maria	869 - X 63 - 4								
		as in a his nan		i ya she y	n an an tha a An tha an tha	ada da seria. Taki	na na sina na sina. Na sina sina sina sina sina sina sina sin	an a					
	a na sana ang sana s Tang sana sang sang sang sang sang sang s	Mini provinstva Sec	a a shina a sana.		an a	waay shiri	an a						
	a defende a ferrale. Normalismente de la composición de la c	ang panahan belaha Mangkaran beraharan	en and the States Market States	e de la factoria Antes de la secon	an a	n an	an da Sura. Manakar		n an Anna An Anna Anna Anna Anna Anna An	- 1992 - 1992 - 1993 - 1994 - 1994			
	in din Album Antonio anto	an tha said an an an Tha an an an an an an an	i di di katalar Mangalari Matsar	an an an Alberta. An an an Alberta an Alberta	na sera de sera en la	a da bahar bahar da Kabupatèn kabupatèn	an an tairte an an An an tairte an						
					a a la constanta de	e de la constante de la consta Este de la constante de la const		an an an Ar	an a ta Rain			-	
्यू हिंदूरी से हैं किसीने के स्थलित जन्म के सिन्द्र के सामग्र संस्थल					د او در کار در دارد. پروند میرد (مود	na an China an Anna An Anna Anna Anna A	and the first second	na shekara shekara Mara shekara s	an a				
and the base of the Alice States and	ana ang ang ang ang Tinggang ang ang ang ang ang ang ang ang a	a de la de la Alexander	e de la desta			a service de partes	10 10 10 20 10 K	a parta da da	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1	2012 - 10 C.	· · · ·		

A. J	226 0		
time:	336.0	ms	
time:	234.2	ms	
time:	233.5	ms	
time:	233.2	ms	
time:	233.5	ms	
Correc	tness o	check:	PASSED

Slow



time:
time:
Correc

long-short	short-long
2) 50ms 60ms 70ms 80ms 90ms 100ms 110ms 120ms 130ms 140ms 150ms	
	Oms 10ms 20ms 30ms 40ms 50ms 60ms 70ms 80ms 90ms 100ms 110ms 120ms 130ms 140ms 150ms

197.5	ms	
147.6	ms	
120.7	ms	
120.6	ms	
120.8	ms	
120.5	ms	
tness o	check:	PASSED
		-

time: 255.3 ms time: 157.8 ms time: 157.1 ms time: 157.1 ms time: 157.1 ms time: 156.9 ms time: 157.5 ms time: 157.1 ms time: 157.2 ms time: 157.2 ms Correctness check: PASSED

1.49x Faster

1.94x Faster





Outline

- Our trace tool ullet
- Potential optimization opportunities of Quicksilver found with the trace tool •
- Exploration of an optimization opportunity \bullet
- Work stealing algorithm among GPU threads
- Summary \bullet



- \bullet stealing
- Pay attention ullet
 - Avoid thread-level spin loop
 - Otherwise, one thread spinning may block the rest in its warp
 - Use __syncwarp()
 - Consider the scenario that not all the threads start at the same time ullet

In Quicksilver, unknown workload imbalance between threads motivates dynamic work



- Each thread starts with its own task, and in each iteration ullet
 - Step 1: works on its task
 - Step 2: decides if it should
 - give another thread its task
 - get another thread's task
 - exit
 - nothing lacksquare
 - Step 3: behaves according to the decision from Step 2 ٠
 - Step 4: ____syncwarp()



- How to decide if the thread is a giver or getter or none?
 - Global variable *done*: number of finished tasks
 - Giver: threads with id in [N *done*, N) && its task still has remaining work
 - Getter: threads with id in [0, N *done*) && its task has no remaining work







How to give or get? Use atomic swap ullet



- giver_count = 0
- getter_count = 0

-1	
-1	
-1	
-1	
-1	
-1	
-1	



How to give or get? Use atomic swap ullet



- \blacktriangleright giver_count = 0
 - getter_count = 0

-1	
-1	
-1	
-1	
-1	
-1	
-1	



How to give or get? Use atomic swap ullet



giver_count = 1

getter_count = 0

-1
-1
-1
-1
-1
-1
-1

Slot 0 gets giver 1's task



How to give or get? Use atomic swap ullet



- $giver_count = 1$
- getter_count = 0

1	
-1	
-1	
-1	
-1	
-1	
-1	
•	



How to give or get? Use atomic swap ullet





How to give or get? Use atomic swap ullet



giver_count = 1

getter_count = 1

-1	
-1	
-1	
-1	
-1	
-1	
-1	
_	
-	



How to give or get? Use atomic swap ullet



giver_count = 1

getter_count = 2



- Each thread starts with its own task, and in each iteration \bullet
 - Step 1: works on its task
 - Step 2: decides if it should
 - give another thread its task
 - get another thread's task
 - exit
 - nothing \bullet
 - Step 3: behaves according to the decision from Step 2 \bullet
 - Step 4: ____syncwarp()

Only if it is NOT trying to get or give

Only try one time of get() or give() in each iteration





Work Stealing Among GPU Threads - Quicksilver



80ms 90ms 100ms 110ms	120ms	



Work Stealing Among GPU Threads - Quicksilver



- Successfully compacted tasks
- Threads started late find no work to do and just exit
- Threads keep trying to get() when only small amount of task remained

100[']ms

110^ms



Outline

- Our trace tool lacksquare
- Potential optimization opportunities of Quicksilver found with the trace tool ●
- Exploration of an optimization opportunity \bullet
- Work stealing algorithm among GPU threads lacksquare
- **Summary**





- ulletoptimization opportunities
- ullet
- A study of Quicksilver revealed two opportunities for optimizations \bullet
 - Imbalance between tasks ullet
 - Imbalance between cases within each iteration \bullet
- Work stealing algorithm on GPU threads seems promising to accelerate the kernel \bullet

Binary instrumentation of GPU device functions within complex kernels reveals potential

Compacting threads with work into small amount of active blocks improves the performance

