Profiling of Data-Parallel Processors

Daniel Kruck

09/02/2014

Outline



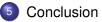
2 Background - GPUs



ProfilerNVIDIA Tools

Lynx

Optimizations



Outline



2 Background - GPUs

Profiler NVIDIA Tools Lynx

4 Optimizations

5 Conclusion

Motivation

Why Data-Parallel Processors?

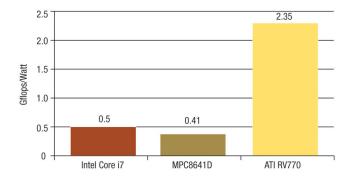


Figure : Energy efficiency comparision: CPU vs GPU [1]

- high energy efficiency
- consume a huge part of the power-budget in HPC

Idea of Data-Parallel Processors



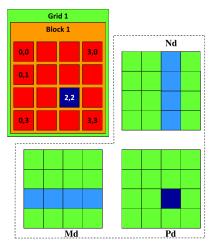


Figure : Idea of data-parallel processors [2]

Figure : Worker thread executes operation on its own element [3]

09/02/2014

Daniel Kruck

Motivation

Why Profiling?

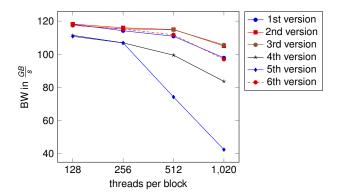


Figure : Device memory bandwidth with respect to threads per block. [4]

- collect runtime information
- optimize objective oriented

~~	10	~	0	~ -	
nc	9/0	121	2	0F	4

Outline

Motivation

2 Background - GPUs

3 Profiler

- NVIDIA Tools
- Lynx

Optimizations

5 Conclusion

x86-CPU and GPU Quiz

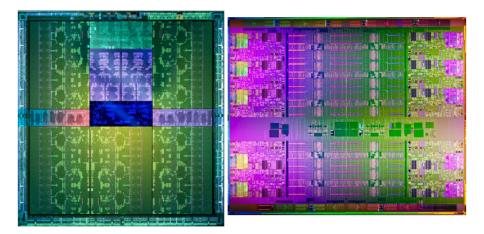


Figure : Which die is the CPU, which one the GPU? [3]

GPU vs CPU

	Tesla K20	Xeon E7-4800 4P
Core count	13 SMs 64/832 (DP), 192/2,496 (SP)	10
Frequency	0.7GHz	2.4GHz
Peak Compute Performance	1,165 GFLOPS (DP) 3,494 GFLOPS (SP)	96 GFLOPS (DP)
Use model	throughput-oriented	latency-oriented
Latency treatment	toleration	minimization
Programming	1000s-10,000s of threads	10s of threads
Memory bandwidth	250 GBytes/sec	34 GByte/s (per P)
Memory capacity	<= 8 GB	up to 2TB
Die size	550mm ²	684 mm ²
Transistor count	7.1 billion	2.3 billion
Technology	28nm	32nm

Figure : GPU vs CPU [3]

09/02/2014	Profiling	Daniel Kruck	9 / 41
03/02/2014	rioning	Danier Kruck	37 - 1

Programming Mode

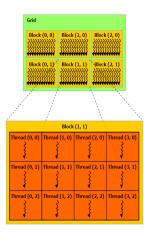


Figure : Programming model

Programming model

- Thread hierarchy: grid, block, warp (usually 32 threads), thread
- Shared memory as scratch-pad memory
- Barrier Synchronization

GPU - Kepler Architecture



Figure : Kepler full chip block [5]

09/02/2014

Profiling

GPU - Kepler Warp Scheduler

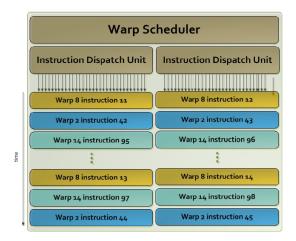


Figure : Kepler warp scheduler [5]

00	00	100	
09/	(02)	20	14

Profiling

GPU-Host Interface

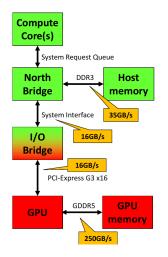


Figure : GPU-Host interface

 in this talk: the red blocks GPU and GPU memory are of interest

- transport of data to GPU memory is expensive
- GPU-GDDR5 memory features high bandwidth

09/02/2014	Profiling	Daniel Kruck	13 / 41
------------	-----------	--------------	---------

Summary

- The cachesize of a GPU is much smaller than of a CPU. Caches are used differently.
- The core-count of GPUs is much higher.
- The communication model between GPU-threads is more relaxed than between CPU-threads. Therefore, there are some differences in the programming model.
- Maximal GPU performance usually decreases the power-budget dramatically. Therefore, GPU applications should be optimized.
- Since there are a lot of mysterious concurrent things going on, runtime information can help to demystify the GPU.

Outline



2 Background - GPUs



Optimizations

5 Conclusion

Definitions (1

Definition

"Application **performance data** is basically of two types: profile data and trace data." [6]

Definition

"**Profile data** provide summary statistics for various metrics and may consist of event counts or timing results, either for the entire execution of a program or for specific routines or program regions." [6]

Definition

"In contrast, **trace data** provide a record of time- stamped events that may include message-passing events and events that identify entrance into and exit from program regions, or more complex events such as cache and memory access events." [6]

Definitions (2)

Definition

"An **event** is a countable activity, action, or occurrence on a device. It corresponds to a single hardware counter value which is collected during kernel execution." [7]

Definition

"A **metric** is a characteristic of an application that is calculated from one or more event values." [7]

NVIDIA Profiling Tools

NVIDIA profiling tools

- *nvprof*: a command line profiler
- Visual Profiler: a tool to visualize performance and trace data generated by the *nvprof*
- **NSight**: a development platform that integrates *nvprof* and *Visual Profiler*
- are based on NVIDIA APIs
 - CUPTI (CUDA Performance Tools Interface): a collection of four APIs, that "enables the creation of profiling and tracing tools" [8]. Through this API metric and event data can be queried, the *nvprof* can be controlled and a lot of other features are exposed.
 - NVML (NVIDIA Management Library): through this library, thermal or power data can be queried.
- are designed to work with NVIDIA GPUs and are easy accessible in a NVIDIA environment

nvprof - Getting Started

help

nvprof ---help

query predefined events

nvprof ---query-events

query predefined metrics

nvprof ---query-metrics

nvprof

example query

nvprof --events elapsed_cycles_sm --profile -from-start-off ./my_application

Invocations	Event Name	Min	Max	Avg
Device "Tesla K2Oc (0)"				
Kernel: Reduction1 kerne	el(int*, int	const *, unsigned	long)	
1 elapse	ed_cycles_sm	99476	99476	99476

Figure : Example output the stated nvprof-command

NSight - Profiling View at a First Glance: Timeline

	s			0.	25 s				0.5	s			0.	75 s			1 s			1.2/5 s
Process 23349																				
Thread 1373886240																				
- Runtime API		cudaM	alloc																	
L Driver API																				
Profiling Overhead																				
[0] Tesla K20c		•			:	1	:	1	:	1	:	•••		:	:	1	- 1			
Context 1 (CUDA)																				
- 🍸 MemCpy (HtoD)																				
– 🍸 MemCpy (DtoH)																				
 Compute 																				
58.2% void Reduction7_kernel <int=32>(int)</int=32>																				
-																				
Y 5.1% Reduction6_kernel(int*, int const *, un																				
Y 5.1% Reduction1_kernel(int*, int const *, un.																				
-																				
└ 🍸 1.8% void reduceSinglePass <unsigned int="</td"><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></unsigned>																				
Y 1.4% void Reduction3_kernel <unsigned <="" int.="" p=""></unsigned>																				
└ 🍸 1.4% void reduceSinglePass <unsigned int="</td"><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></unsigned>																				
└ \ 1.2% void reduceSinglePass <unsigned int="</td"><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></unsigned>																				
Y 1.2% void Reduction3_kernel <unsigned <="" int.="" p=""></unsigned>																				
└ 〒 1.1% void reduceSinglePass <unsigned int="</td"><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></unsigned>																				
Y 1.1% void Reduction3_kernel <unsigned <="" int.="" p=""></unsigned>																				
Y 1.0% void Reduction3_kernel <unsigned <="" int.="" p=""></unsigned>																				
– 🍸 0.0% memset (0)																				
Streams																				
- Default																				

Figure : Nsight profiling view: timeline

00	00	00	4.4
09/	(02)	20	14

Profiling

NSight - Detection of Obvious Mistakes - Occupancy

Definition

Occupancy is the ratio between active warps and the maximum amount of active warps.

Compute						G	obal Load Efficien	ncy	10	076
- Y 19.3% Reduction						G	obal Store Efficier	ncy	<u>å</u> 12	.5%
- 7 29.3% Reduction						w	arp Execution Eff	iciency	<u>4</u> 68	.2%
- Y 21.3% Reduction			_			N	on-Predicated Wa	arp Executi	on E 🚯 67	.1%
- 7 10.3% Reduction							ipancy			
└ 〒 10.0% Reduction						A	hieved		<u>å</u> 24	.4%
- Y 1.0% void Reduct.						T	eoretical		25	96
- Y 0.02 void Reduct						U	miter		Ble	ock Siz
	unsigi (a) GPU Utilization I The kernel has a block	size of 32 threads.	This block size is							
ction0a_kernel(int*, int*, nel Perfoance Limiter	unsigi The kernel has a block	size of 32 threads. te up to 16 blocks o each SM. Chart "Va SM.	This block size i in each SM. Bec arying Block Size	ause each bloc " below shows l	k úses 1 wa now changir	rp to exec Ig the bloc	ute the block's 3 k size will change	2 threads, I the numb	the kernel	is s that
ction0a_kernel(int*, int*, nel Perfoance Limiter	GPU Utilization I The kernel has a block simultaneously execut using only 16 warps on can execute on each s	size of 32 threads. te up to 16 blocks o each SM. Chart "Va SM.	This block size i in each SM. Bec arying Block Size	ause each bloc " below shows l : to increase the	k uses 1 wa now changin e number of	rp to exec ig the bloc warps tha	ute the block's 3 k size will change	2 threads, I e the numb each SM.	the kernel er of warp:	is s that <u>More.</u>
ction0a_kernel(int*, int*, nel Perfoance Limiter	GPU Utilization I The kernel has a block simultaneously execut using only 16 warps on can execute on each 9 Optimization: Increase Variable Occupancy Per SM	size of 32 threads. te up to 16 blocks o each SM. Chart "Ve SM. the number of three	This block size is in each SM. Bec arying Block Size ads in each block	ause each bloc " below shows l : to increase the	k uses 1 wa now changin e number of	rp to exec ig the bloc warps tha	ute the block's 3 k size will change t can execute on e	2 threads, I e the numb each SM.	the kernel er of warp:	is s that <u>More.</u>
ction0a_kernel(int*, int*, nel Perfoance Limiter	GPU Utilization I Me kernel has a block simultaneously execut using only 16 warps on can execute on each 5 Optimization: Increase Variable	size of 32 threads. te up to 16 blocks o each SM. Chart "Ve SM. the number of three	This block size is in each SM. Bec arying Block Size ads in each block	ause each bloc " below shows l : to increase the	k uses 1 wa now changin e number of	rp to exec ig the bloc warps tha	ute the block's 3 k size will change t can execute on e	2 threads, 1 2 the numb each SM. ocks) Block	the kernel er of warp:	is s that <u>More.</u>
el Compute		size of 32 threads. te up to 16 blocks o each SM. Chart "Ve SM. the number of three	This block size is in each SM. Bec arying Block Size ads in each block Theoretical	ause each bloc " below shows l c to increase the Device Limit	k uses 1 wa now changin e number of	rp to exec ig the bloc warps tha	úte the block's 3: k size will change t <i>can execute on e</i> 1,1] (1048576 blo	2 threads, I e the numb each SM. ocks) Block	the kernel er of warp:	is s that <u>More.</u>
tion0a_kernel(int*, int*, hel Perfoance Limiter () hel Latency hel Compute hel Memory () hory Access Pattern ()	GPU Utilization I GPU Utilization I The kernel has a block simultaneously execute or each 1 Optimization: Increase Variable Occupancy Per SM Active Blocks	size of 32 threads. te up to 16 blocks o each SM. Chart "Ve SM. Achieved	This block size is in each SM. Bec arying Block Size ads in each block Theoretical 16	ause each bloci * below shows l to increase the Device Limit 16	k uses 1 wai now changin e number of Grid Size:	rp to exec ig the bloc warps tha	úte the block's 3: k size will change t can execute on e 1,1] (1048576 bk 5 6 7 8 9 21 28 35	2 threads, I e the numb each SM. ocks) Block	the kernel er of warp: Size: (32 2 13 14 1 19 56	is s that <u>More.</u> (1,1] (3
tion0a_kernel(int*, int*, hel Perfoance Limiter () hel Latency hel Compute hel Memory () hory Access Pattern ()	GPU Utilization I The kernel has a block smutaneously execute an and can execute on each i optimization: Increase Variable Cocupancy Per SM Active Blocks Active Warps	size of 32 threads. te up to 16 blocks o each SM. Chart "Ve SM. Achieved	This block size is in each SM. Bec arying Block Size ads in each block Theoretical 16 16	ause each bloci * below shows l to increase the Device Limit 16 64	k uses 1 wai now changin e number of Grid Size:	p to exec g the bloc warps that [1048576, 2 3 4 14	úte the block's 3; k size will change t can execute on e 1,1] (1048576 bk 5 6 7 8 9 21 28 35 768 1024	2 threads, i a the numb each SM. ocks) Block 10 11 12 42 4 1280 153	the kernel er of warp: Size: (32 2 13 14 1 19 56	is s that (1,1) (3 15 16 684

Figure : Occupancy example: kernel block size to small

NSight - Detection of Obvious Mistakes - **Branch** Divergency

Definition

Branch divergency on a GPU refers to divergent control-flow for threads within a warp. [9]

source of branch divergence

if (tid % 2 == 0)
sPartials[tid] += sPartials[tid];



Figure : Example: branch divergence

NSight - Detection of Obvious Mistakes - Coalesce Access

Definition

Coalesce access refers to to the aligned consecutive memory access pattern of an active warp.

source of inefficient access pattern

Global Load Efficiency	100%
Global Store Efficiency	12.5%

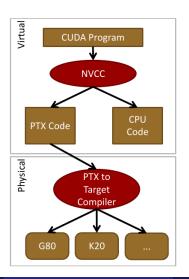
Figure : Example: global store inefficiency

09/02/2014	Profiling	Daniel Kruck	24 / 41

PAPI & TAU

• PAPI (Performance Application Programing Interface)

- + has a broad userbase
- + gives access to common hardware counters through a consistent interface
- + portable code
- is based on PAPI CUDA component
- requires CUPTI-enabled driver
- TAU (Tuning and Analysis Utilities)
 - + well-known to HPC developers consistent interface
 - + portable code
 - TAU relies on CUDA library wrapping just like PAPI



- NVCC separates PTX from HOST code
- PTX code is later on translated to device code
- the compilation of PTX code can be ahead-of-time (AOT) or just-in-time (JIT)
- PTX code provides an opportunity for a custom instrumentation

09/02/2014

Profiling

Lynx

- dynamic instrumentation +
- transparent, selective +

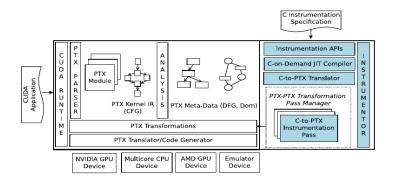


Figure : Lynx software architecture [10]

00/	20	100		
J9/	02	20	14	

Lynx - Instrumentation Specifications

Kernel Runtimes and Thread Block-SM Mapping

unsigned long start, stop;

insert instrumentation at the ON KERNEL ENTRY: beginning of every kernel start = clockCounter(); syncThreads(); insert instrumentation at the ON KERNEL EXIT: 4 end of everv kernel syncThreads(); stop = clockCounter(); if(threadIndexX() = 0)globalMem[blockId() * 2] = stop - start; globalMem[blockId() * 2 + 1] = smId();}

- + fine grain profiling
- + selective
- + transparent

Figure : Lynx instrumentation specifications [10]

09/	02	/20	14
00,	~		

+ online profiling

Features	CUPTI	Lynx
Transparency (No Source Code Modifica- tions)	Yes	Yes
Support for Selective Online Profiling	No	Yes
Customization (User-Defined Profiling)	No	Yes
Ability to Attach/Detach	No	Yes
Support for Comprehensive Online Profiling	No	Yes
Support for Simultaneous Profiling of Multiple Metrics	No	Yes
Native Device Execution	Yes	Yes

Figure : Distinctive features of lynx [10]

- NVIDIA tools:
 - + easy accessible in NVIDIA environment
 - + common errors can be automatically detected with the automated analysis engine
 - no fine-grain profiling
 - not as selective and customizable as LYNX
- PAPI & TAU:
 - familiar to PAPI or TAU users
 - are basically wrapper libraries on NVIDIA APIs and therefore have the same strengths and weaknesses
- Lynx
 - transparent and highly selective instrumentation
 - + not restricted to NVIDIA GPUs through the Ocelot-Cross-Compiler
 - + online profiling possible
 - not pre-installed in NVIDIA environments ;)

Outline



2 Background - GPUs

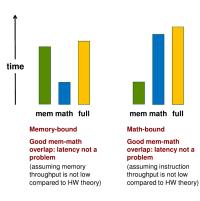
Profiler NVIDIA Tools Lynx

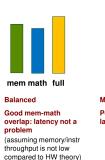
Optimizations

5 Conclusion

Optimizations

Detect Bottleneck with math-only or memory-only Kernels







mem math full

Memory and latency bound

Poor mem-math overlap: latency is a problem

- Profile the global memory transactions for the memory-only kernel.
- Profile the register-count for the math-only kernel.

09/02/2014

Profiling

Daniel Kruck

Latency Profiling

- NVIDIA Tools : try to isolate a block.
- Lynx : just use fine-grained profiling.

nvprof latency profiling example

```
nvprof ---aggregate-mode off
---events elapsed_cycles_sm
---profile-from-start-off ./reduction
```

Bottleneck Detected! What to do Next?

math-bound

- are there divergent branches?
- simplify indexing math?
- are there sequences of same operations? (pipeline stall)

memory-bound

- profile access pattern.
- look out for opportunities to improve occupancy.
- Iatency-bound
 - is there a chance to optimize thread synchronization?
 - is there a chance to increase independent instructions?

Outline

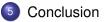


2 Background - GPUs

Profiler NVIDIA Tools

Lynx

Optimizations



Conclusion

Data-Parallel Processors

- + high power efficiency
- + are commonly used in the field of HPC
- + fun to play with
- complex runtime behaviour
- complex programming model, differs from CPU model

Profiling

- Native NVIDIA tools
 - + easy accessible
 - + fast detection of common mistakes
- Alternatives like lynx showcase interesting new features like
 - online profiling
 - fine grain profiling

Quo Vadis, Data-Parallel Profiling?

- the amount of devices in supercomputers increases
- the energy-budget is becoming more and more the limiting factor

Future of Data-Parallel Profiling

Is there a shift towards energy profiling of entire systems?

Discussion



Figure : Source: "The Internet" ;) - Questions??

09/02/2014

For Further Reading



Scott Theiret Anne Mascarin.

Cpus and gpus vie for new signal and image processing roles. http:

//www.cotsjournalonline.com/articles/view/101617, 2010.

- Guillermo Marcus.

Gpu computing, 2012.

Holger Froening. Gpu lecture, 2013.

Nicholas Wilt.

The cuda handbook. 2013.

For Further Reading II

Nvidia.

Whitepaper: Nvidia's next generation cuda compute architecture: Kepler gk110.

Shirley Moore, David Cronk, Felix Wolf, Avi Purkayastha, Patricia Teller, Robert Araiza, Maria Gabriela Aguilera, and Jamie Nava. Performance profiling and analysis of dod applications using papi and tau.

In Users Group Conference, 2005, pages 394–399. IEEE, 2005.

NVIDIA.

Profiling user's guide.

http://docs.nvidia.com/cuda/profiler-users-guide, 2013.

For Further Reading II

NVIDIA. Cupti, 2013.



Jennifer Hohn.

Optimizing application performance with cuda profiling tools, 2012.

Naila Farooqui, Andrew Kerr, Greg Eisenhauer, Karsten Schwan, and Sudhakar Yalamanchili.

Lynx: A dynamic instrumentation system for data-parallel applications on gpgpu architectures.

In Performance Analysis of Systems and Software (ISPASS), 2012 IEEE International Symposium on, pages 58–67. IEEE, 2012.