
Profiling
and
Performance
Engineering
5月16日 20:30
学生科创中心（H2 一楼）
主讲人：谢志强

Profiling

In short, **Profiling** is to identify where your code is slow

Program analysis tools are extremely important for **understanding program behavior**. *Computer architects* need such tools to evaluate how well programs will perform on new architectures. ***Software writers*** need tools to analyze their programs and identify critical sections of code. *Compiler writers* often use such tools to find out how well their instruction scheduling or branch prediction algorithm is performing...
— ATOM, PLDI, '94

Profiling

"Premature optimization is the root of all evil"

-Donald Knuth

- Should focus on optimizing **hotspots**
- It's not possible to go through the codebase for optimizing in most cases

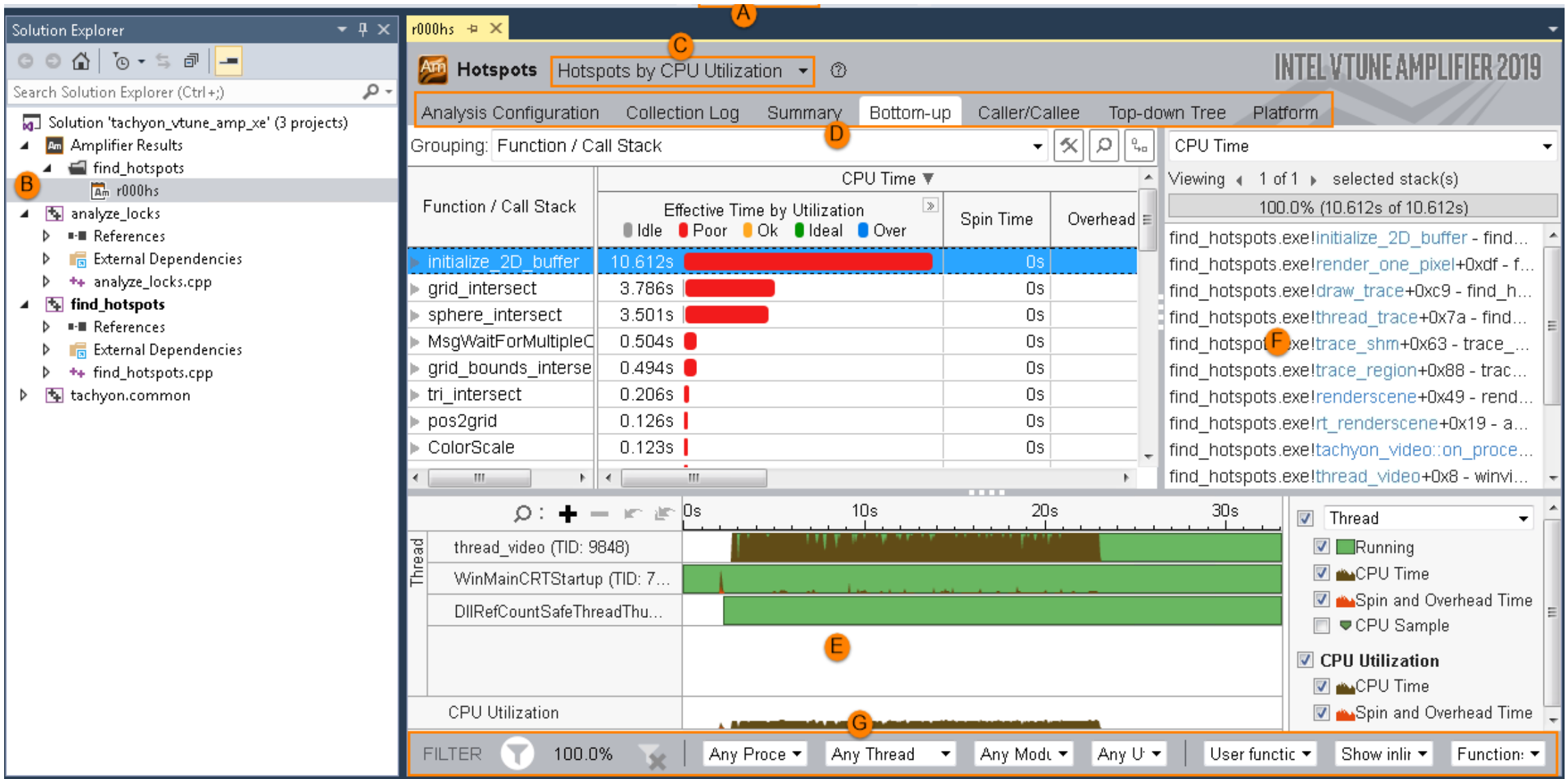
Bottleneck

- **Compute-bound**
  - CPU, GPU Streaming Processors, etc
- **Memory-bound**
  - Intensive data access, cache efficiency
- I/O bound:
  - network, disk
- Idle due to schedule issues
  - Processors wait for task from user or scheduler

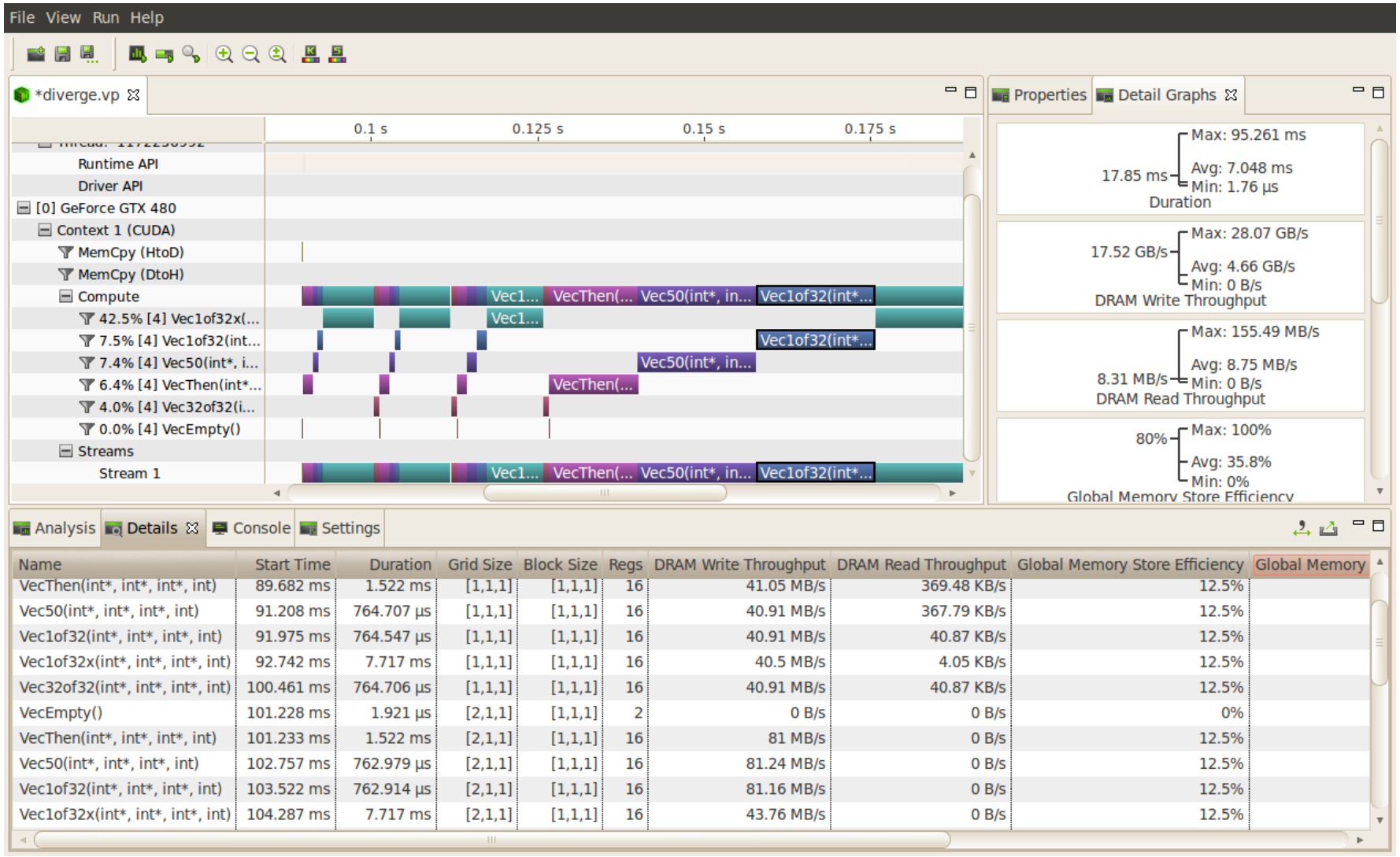
Profilers

- Manual Instrumentation
  - Print performance information manually designed by yourself
- Static Instrumentation
  - Gprof, ATOM, Etch, etc
- Performance Counter
  - OProfile, perf
- Integrated Profiler
  - Intel® VTune™ Amplifier
  - NVIDIA Visual Profiler

Intel® VTune™ Amplifier



NVIDIA Visual Profiler



Name	Start Time	Duration	Grid Size	Block Size	Regs	DRAM Write Throughput	DRAM Read Throughput	Global Memory Store Efficiency	Global Memory
VecThen(int*, int*, int*, int)	89.682 ms	1.522 ms	[1,1,1]	[1,1,1]	16	41.05 MB/s	369.48 KB/s	12.5%	
Vec50(int*, int*, int*, int)	91.208 ms	764.707 µs	[1,1,1]	[1,1,1]	16	40.91 MB/s	367.79 KB/s	12.5%	
Vec1of32(int*, int*, int*, int)	91.975 ms	764.547 µs	[1,1,1]	[1,1,1]	16	40.91 MB/s	40.87 KB/s	12.5%	
Vec1of32x(int*, int*, int*, int)	92.742 ms	7.717 ms	[1,1,1]	[1,1,1]	16	40.5 MB/s	4.05 KB/s	12.5%	
Vec32of32(int*, int*, int*, int)	100.461 ms	764.706 µs	[1,1,1]	[1,1,1]	16	40.91 MB/s	40.87 KB/s	12.5%	
VecEmpty()	101.228 ms	1.921 µs	[2,1,1]	[1,1,1]	2	0 B/s	0 B/s	0%	
VecThen(int*, int*, int*, int)	101.233 ms	1.522 ms	[2,1,1]	[1,1,1]	16	81 MB/s	0 B/s	12.5%	
Vec50(int*, int*, int*, int)	102.757 ms	762.979 µs	[2,1,1]	[1,1,1]	16	81.24 MB/s	0 B/s	12.5%	
Vec1of32(int*, int*, int*, int)	103.522 ms	762.914 µs	[2,1,1]	[1,1,1]	16	81.16 MB/s	0 B/s	12.5%	
Vec1of32x(int*, int*, int*, int)	104.287 ms	7.717 ms	[2,1,1]	[1,1,1]	16	43.76 MB/s	0 B/s	12.5%	

Metrics

From perf:

- **cpu-cycles OR cycles**
- **instructions**
- cache-references
- **cache-misses**
- branch-instructions OR branches
- **branch-misses**
- bus-cycles
- stalled-cycles-frontend OR idle-cycles-frontend
- stalled-cycles-backend OR idle-cycles-backend
- ref-cycles
- …

From nvprof:

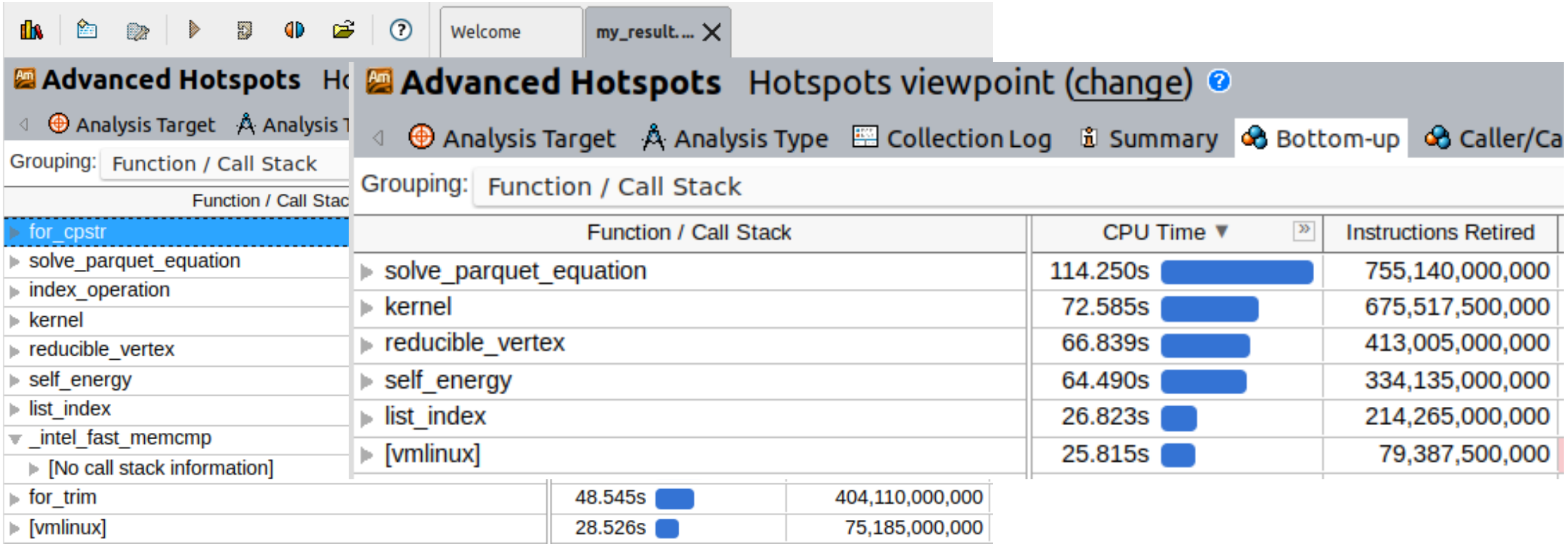
- achieved_occupancy
- branch_efficiency
- dram_utilization
- gld_efficiency
- inst_issue
- ipc
- sm_efficiency
- warp_execution_efficiency
- …

Also, the theoretical peak performance

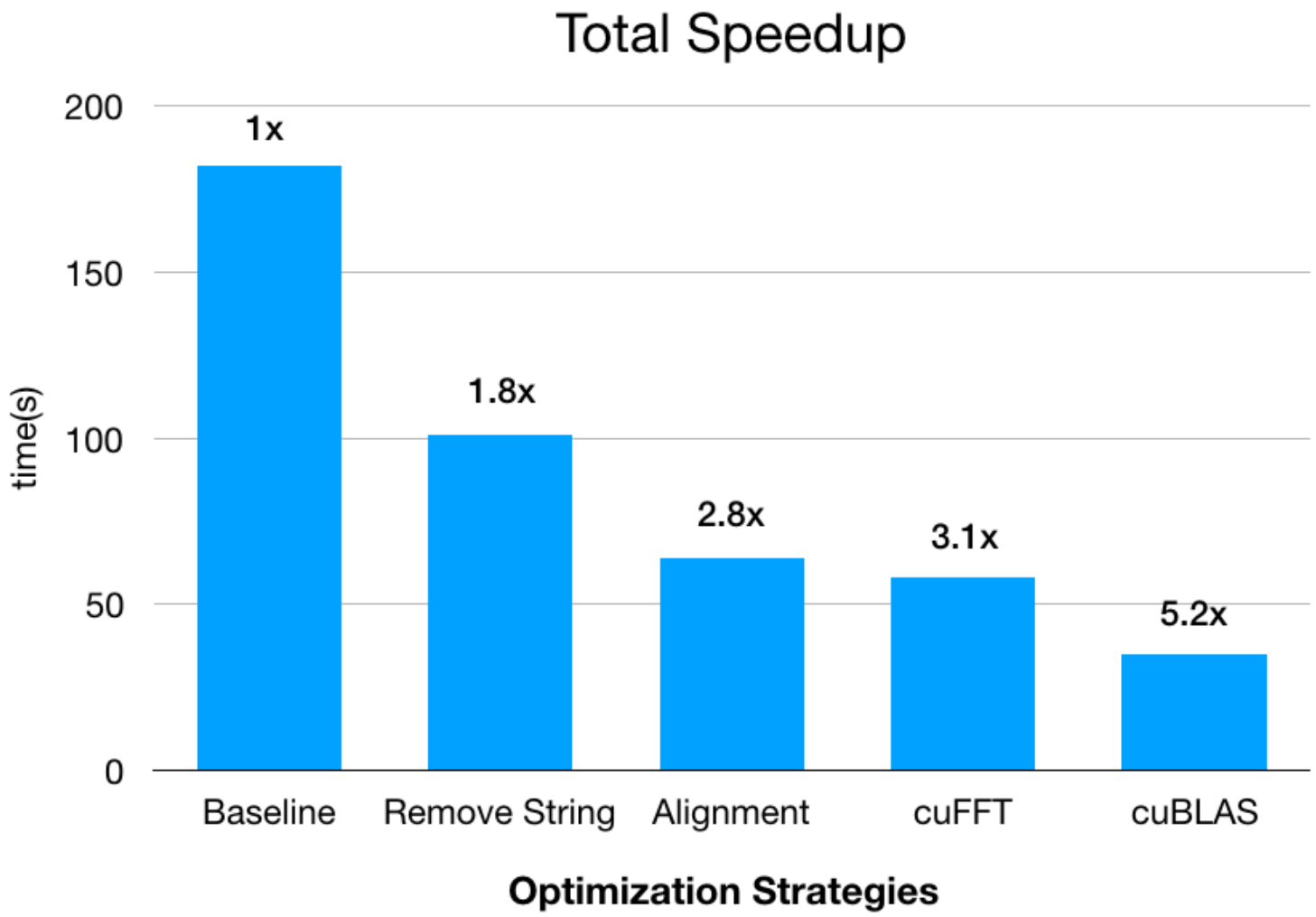
Case study: Victory

Use integer tags to replace strings.

Speed up is about 2x.

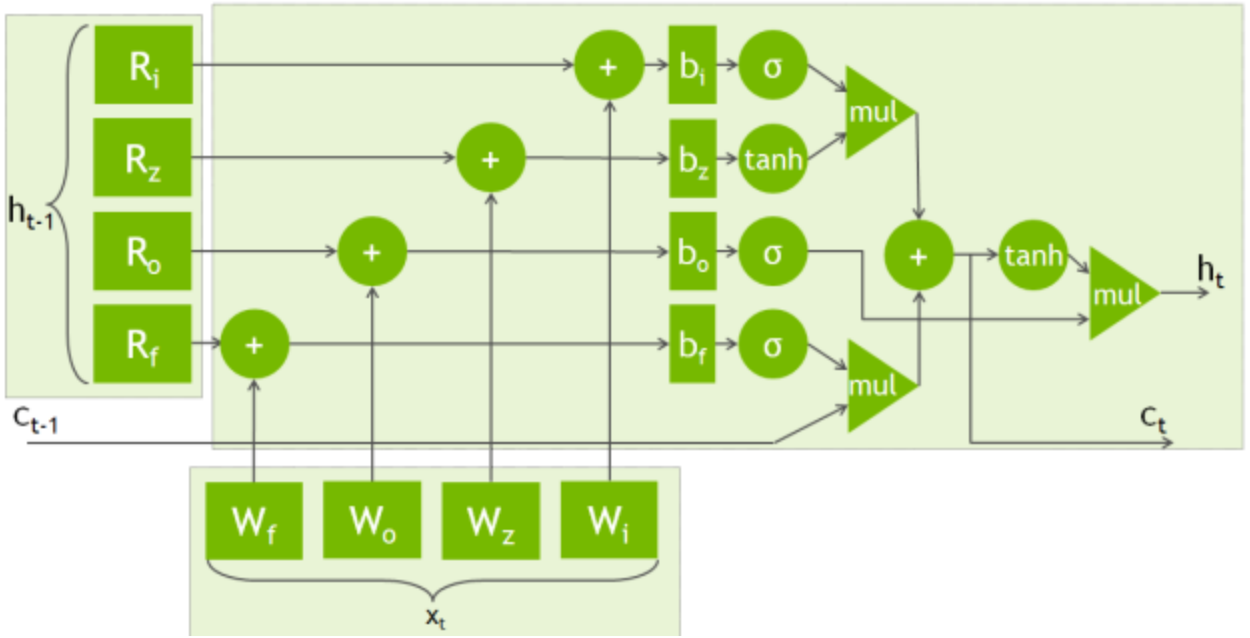


Case study: Victory



- Better implementation
- Heterogeneous processing
- Amdahl's law

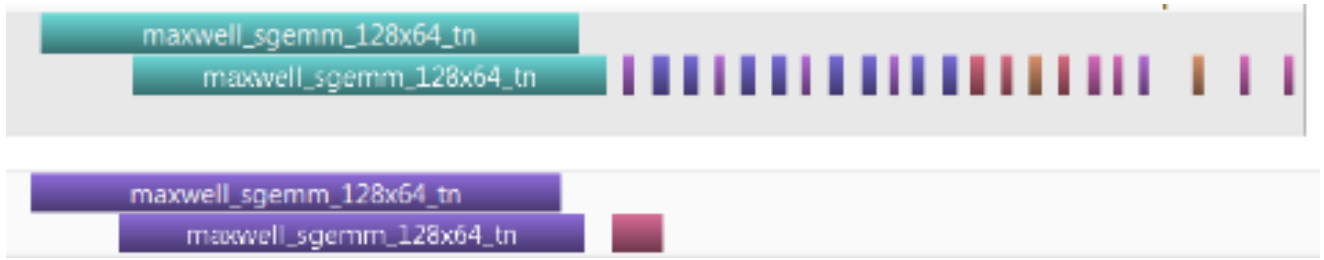
Case study: LSTM on GPU

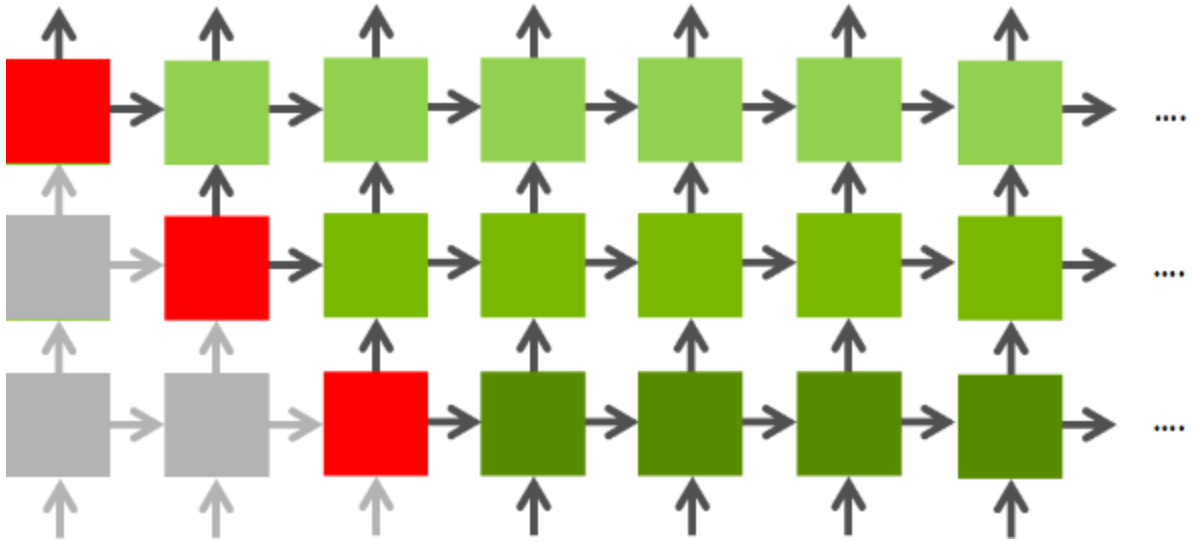


Naïve implementation:
0.4 TFLOPS
Larger matrix operations:
0.8 TFLOPs
Stream matrix operations:
1.12 TFLOPs

Run on NVIDIA M40 GPU with a host CPU Intel Xeon CPU E5-2690 v2 @ 3.00GHz

Case study: LSTM on GPU





Stream matrix operations:
1.12 TFLOPs
Fused point-wise operations:
2.24 TFLOPs
Overlapping layers and some other minor improvements
4.4 TFLOPs

The theoretical peak performance of M40
5.8 TFLOPS

Thank you for your attending!
Q&A