

# OpenCL for NVIDIA GPUs

Chris Lamb  
NVIDIA



# Outline

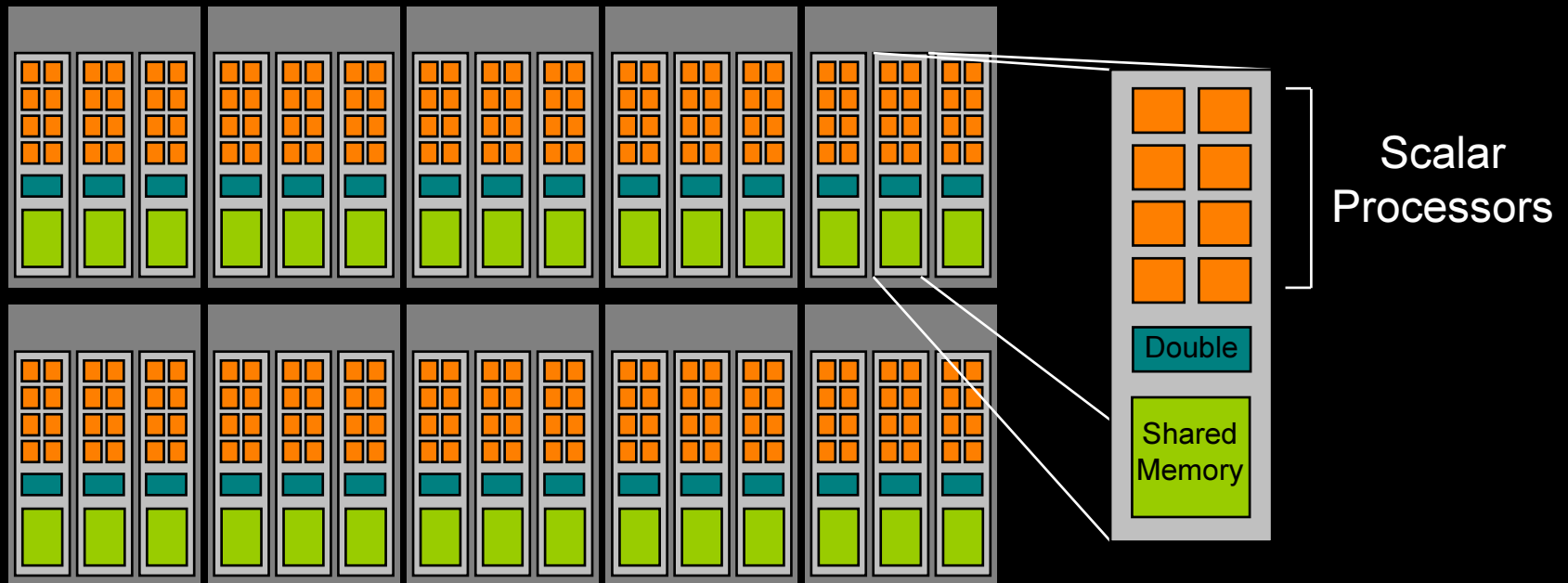
- **OpenCL on NVIDIA GPUs**
- **Best Practices**
- **Demos**

# OpenCL on NVIDIA GPUs

- **OpenCL is a great match**
  - **Direct correlation to the HW execution model**
  - **Direct correlation to the HW memory model**
  - **Direct correlation to the HW synchronization model**
- **Based on NVIDIA's mature GPU computing infrastructure**
  - **Compiles to PTX ISA**
  - **Profiling signals and instrumentation**
  - **Drivers and full SDK available since May**
  - **Available to all professional developers/researchers today**
- **Supported on hundreds of millions of CUDA-enabled GPUs already in the market**

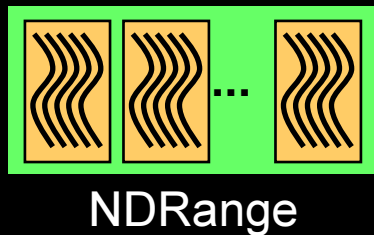
# NVIDIA 10-Series Architecture

- **240 Streaming Processor (SP) cores** execute HW threads
- **30 Streaming Multiprocessors (SMs)** each of which contain:
  - 8 multi-threaded streaming processors
  - 2 Special Function Units (SFUs)
  - 1 **double precision unit**
  - **Shared memory** that enables work-item cooperation

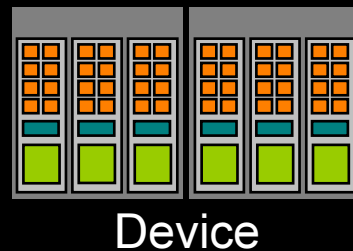
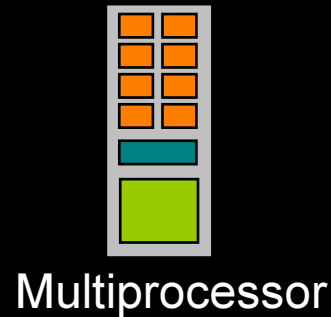


# OpenCL Execution Model on NVIDIA

## Software



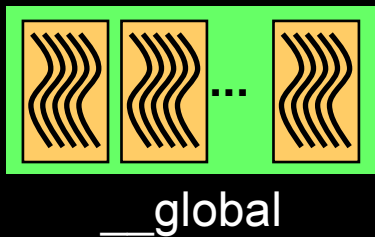
## Hardware



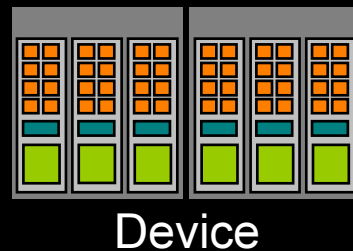
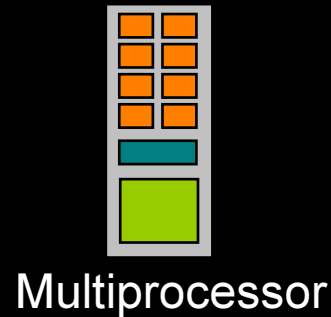
- Work-items are executed by streaming processors
- Maps directly to HW managed threads
- Work-groups are executed on multiprocessors
- They do not migrate
- Several concurrent work-groups can reside on one multiprocessor - limited by multiprocessor resources
- A kernel is launched as an N-D Range of work-groups
- Work-groups are dynamically load-balanced by HW scheduler

# OpenCL Memory Model on NVIDIA

## Software




## Hardware



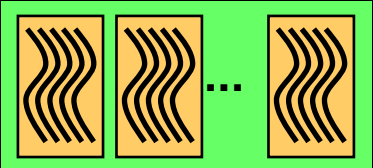
- Each hardware thread has a dedicated \_\_private region for stack
- Each multiprocessor has dedicated storage for \_\_local memory and \_\_constant caches
- Work-items running on a multiprocessor can communicate through \_\_local memory
- All work-groups on the device can access \_\_global memory
- Atomic operations allow powerful forms of global communication

# OpenCL Synchronization on NVIDIA

## Software

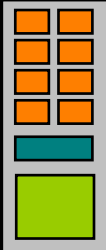
  
mem\_fence()  
atom\_\*(())

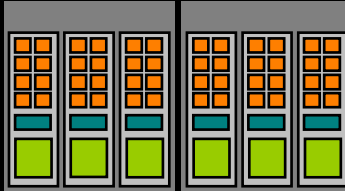
  
barrier()  
work\_group\_copy()

  
EnqueueNDRange  
cl\_event

## Hardware

  
Scalar  
Processor

  
Multiprocessor

  
Device

- Independent atomic operations and memory system control

- Write collective operations in a familiar C-style

- Single instruction fast barrier support directly in HW

- Collective operations leverage the entire multi-processor

- Direct HW support for scheduling NDRange grids

- Direct HW support for scheduling enqueued commands using cl\_events

## Outline

- OpenCL on NVIDIA GPUs
- **Best Practices**
- Demos

# Scalar Architecture

- **NVIDIA GPUs have a scalar architecture**
  - **Use vector types in OpenCL for convenience, not performance**
  - **Generally want more work-items rather than large vectors per work-item**
- **Optimize performance by overlapping memory accesses with HW computation**
  - **High arithmetic intensity programs (i.e. high ratio of math to memory transactions)**
  - **Many concurrent work-items**

# Take Advantage of `__local` Memory

- Hundreds of times faster than `__global` memory
- Work-items can cooperate via `__local` memory
  - `barrier()` only needs `CLK_LOCAL_MEM_FENCE`, which is much lower overhead
- Use it to manage locality
  - Stage loads and stores in shared memory to optimize reuse

# Optimize Memory Access

- **Assess locality of \_\_global memory access patterns**
  - HW coalescing of accesses within 128-byte memory blocks
  - 1<sup>st</sup> Order performance effect
- **Optimize for spatial locality of accesses in cached texture memory (OpenCL Images)**
  - Image reads may benefit from processing as 2D blocks
  - Experiment with work-group aspect ratio to discover what's best
- **Let OpenCL allocate memory optimally**
  - `CL_MEM_ALLOC_HOST_PTR`
  - The implementation can optimize alignment and location
  - Can still get access for the host via `clEnqueueMap{Buffer|Image}`

# Transfer/Compute Overlap

- **Separate command queues can always overlap**
  - Can use this to overlap transfer and compute
  - Generally best when transfer and compute time is balanced
  - Most useful when data has high reuse
- **Or directly pass `ALLOC_HOST` memory to kernel**
  - Uses GPU's latency hiding to ensure maximal bus usage
  - Generally best when data has low/no reuse
  - No events needed to synchronize between copy and kernel

# Use Parallelism Efficiently

- **Partition your computation to keep the GPU multiprocessors equally busy**
  - Many work-items, many work-groups
  - work-groups  $\gg$  number of Compute Units
  - 256-512 work-items per work-group a good target
- **Keep resource usage low enough to support multiple active work-groups per multiprocessor**
  - Registers, `__local` memory can reduce available parallelism
  - Use Occupancy Calculator tool to help estimate

# Math Library and Compiler

- **Use native\_\* and half\_\* math functions where possible**
  - Many have a direct mapping to hardware ISA
  - Can be orders of magnitude faster than higher precision variants
  - *Note that half\_\* functions do not mean the FP16 type extension*
- **Use the -cl-mad-enable compiler option**
  - Permits use of FMADs, which can lead to large performance gains
- **Investigate using the -cl-fast-relaxed-math compiler option**
  - enables many aggressive compiler optimizations

## Outline

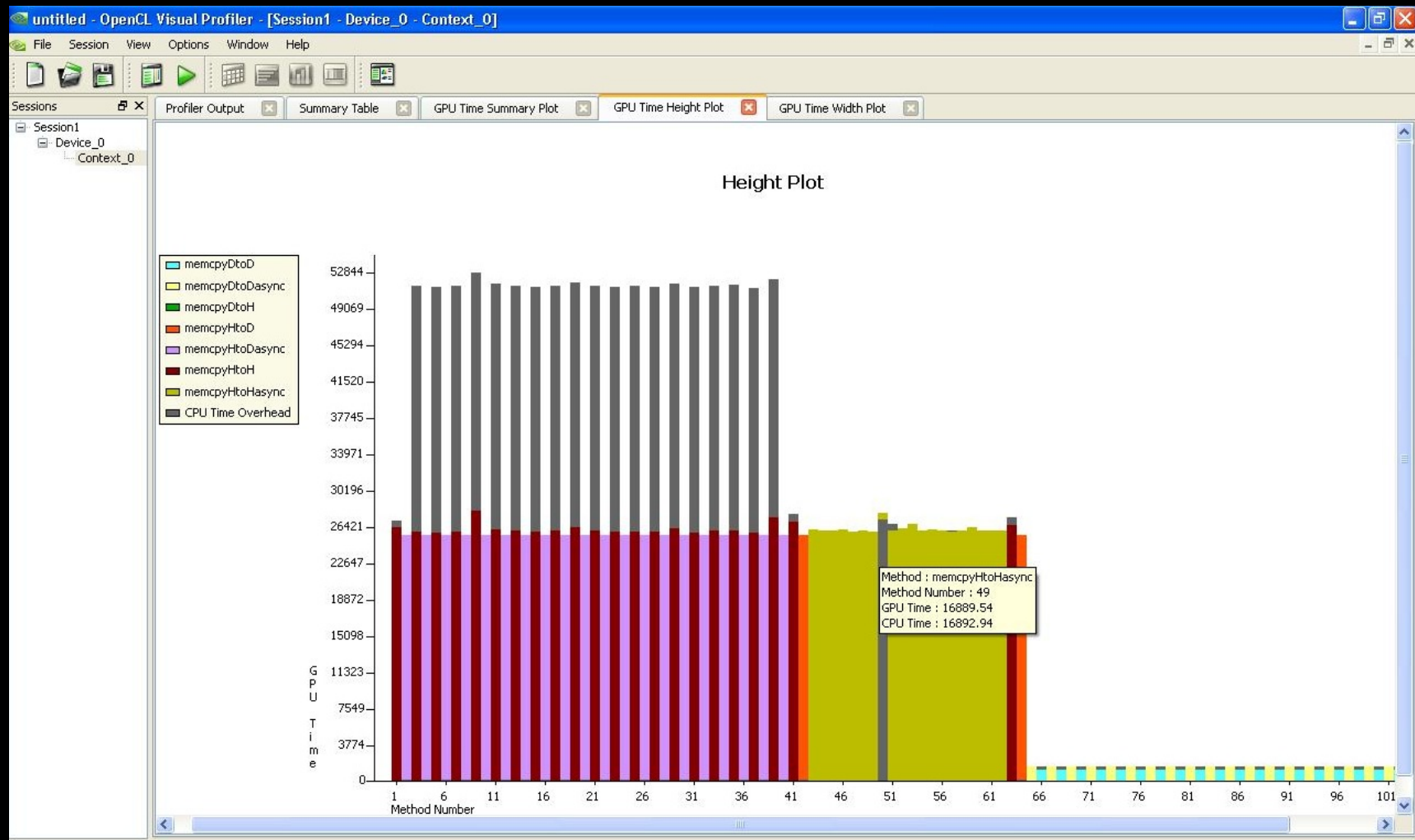
- **OpenCL on NVIDIA GPUs**
- **Best Practices**
- **Demos**

# OpenCL Profiler Overview

- **Profiler facilitates analysis and optimization of OpenCL programs by:**
  - **Reporting hardware counter values:**
    - **Number of various bus transactions**
    - **Branches**
    - **Effective Parallelism**
    - **Etc.**
  - **Computing per kernel statistics:**
    - **Effective instruction throughput**
    - **Effective memory throughput**
  - **Visually displaying time spent in various GPU calls**
- **Requires no instrumentation of the source code**

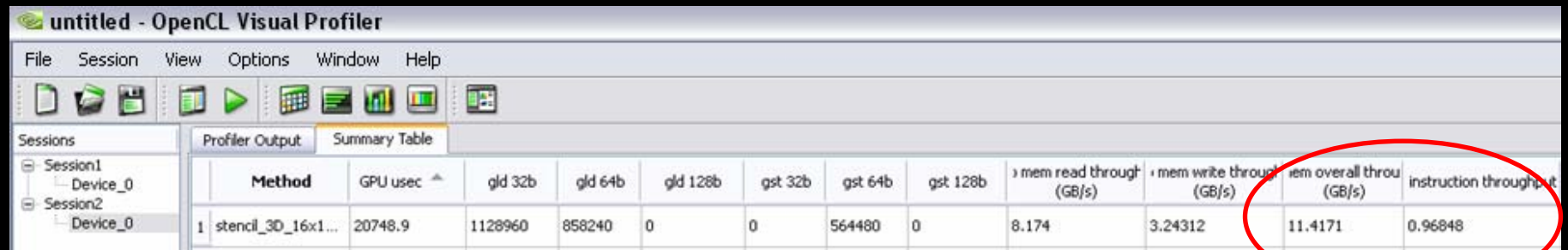
# OpenCL Profiler Example

## Time profile of GPU operations



# OpenCL Profiler Sample Uses

- Determining whether kernel performance is bound by instruction or memory throughput
- Determining whether performance is limited by kernel execution or data transfer times
- Determining percentage of the application time spent in each kernel



The screenshot shows the OpenCL Visual Profiler interface. The main window displays a summary table of kernel performance metrics. The table has columns for Method, GPU usec, and various throughput metrics. The 'mem overall throu (GB/s)' and 'instruction throughput' columns are circled in red.

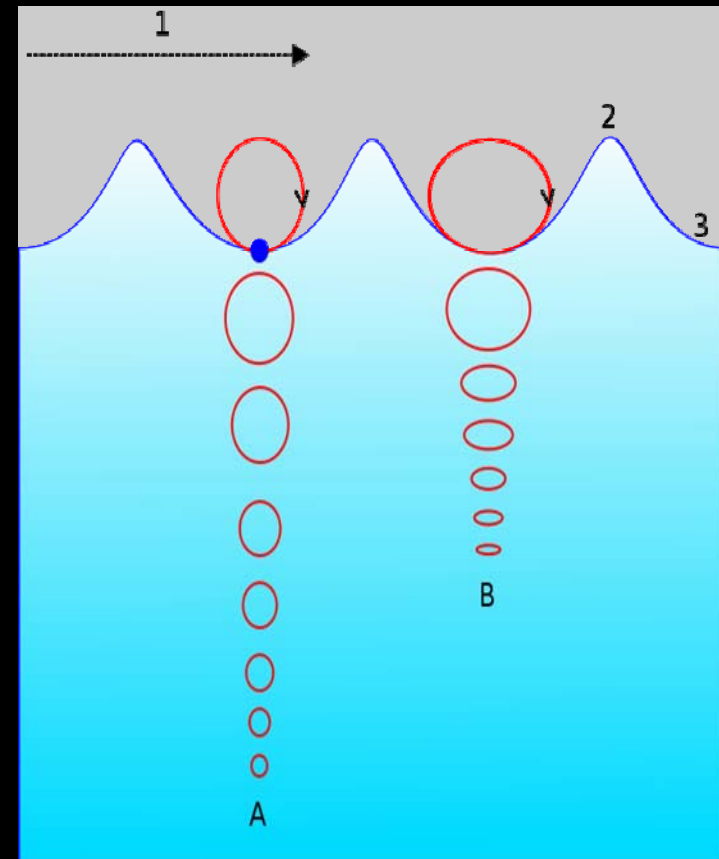
Method	GPU usec	gld 32b	gld 64b	gld 128b	gst 32b	gst 64b	gst 128b	mem read through (GB/s)	mem write through (GB/s)	mem overall throu (GB/s)	instruction throughput
1 stencil_3D_16x1...	20748.9	1128960	858240	0	0	564480	0	8.174	3.24312	11.4171	0.96848

# Ocean Simulation Overview

- **Based on Jerry Tenssendorf's paper "Simulating Ocean Water"**
  - **Statistic based, not physics based**
  - **Generate wave distribution in frequency domain, then perform inverse FFT**
  - **Widely used in movie CGIs since 90s, and in games since 2000s**
- **In movie CGI: The size of height map is large**
  - **Titanic, 2048x2048**
  - **Water World, 2048x2048**
- **In games: The size of height map is small**
  - **Crysis, 64x64**
  - **Resistance 2, 32x32**
  - **All simulated on CPU (or Cell SPE)**

# The Algorithm: Wave Composition

- The ocean surface is composed by enormous simple waves
- Each sine wave is a hybrid sine wave (Gerstner wave)
  - A mass point on the surface is doing vertical circular motion



# Performance Issues

- **The simulation is required to generate the displacement map in real-time**
- **Computing FFT on CPU becomes the bottleneck when the displacement map gets larger**
  - Larger texture also takes longer time on CPU-GPU data transfer
  - However, large displacement map is a must-have for detailed wave crests
- **GPU computing is really good at FFT**
  - Multiple 512x512 transforms can be performed in trivial time on high-end GPUs
  - Demo uses multiple 1024x1024 transforms, clearly affordable for high quality real-time rendering

# Ocean Simulation Demo



[nvidia.com/opengl](http://nvidia.com/opengl)

[developer.nvidia.com/page/registered\\_developer\\_program.html](http://developer.nvidia.com/page/registered_developer_program.html)



# GPU Technology Conference

Sept 30 – Oct 2, 2009 – The Fairmont San Jose, California

*The most significant event in 2009 dedicated to application development on the GPU*

- Learn about the seismic shifts happening in computing
- Preview disruptive technologies and emerging applications to stay ahead of imminent trends
- Get tools/techniques to impact mission critical projects now
- Network with experts and peers from across several industries

## Opportunities:

- Call for Submissions open for talks, tutorials, panels, birds of a feather, posters and moderated roundtables
- Sponsors / Exhibitors have a variety of options to reach influential decision makers
- Startup Showcase  
Present your company and technology to potential investors



Event focused on developers, engineers, researchers, senior executives, venture capitalists, press and analysts

[nvidia.com/gtc](http://nvidia.com/gtc)

© 2009 NVIDIA CORPORATION

