Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

GPU Profiling with AMD CodeXL

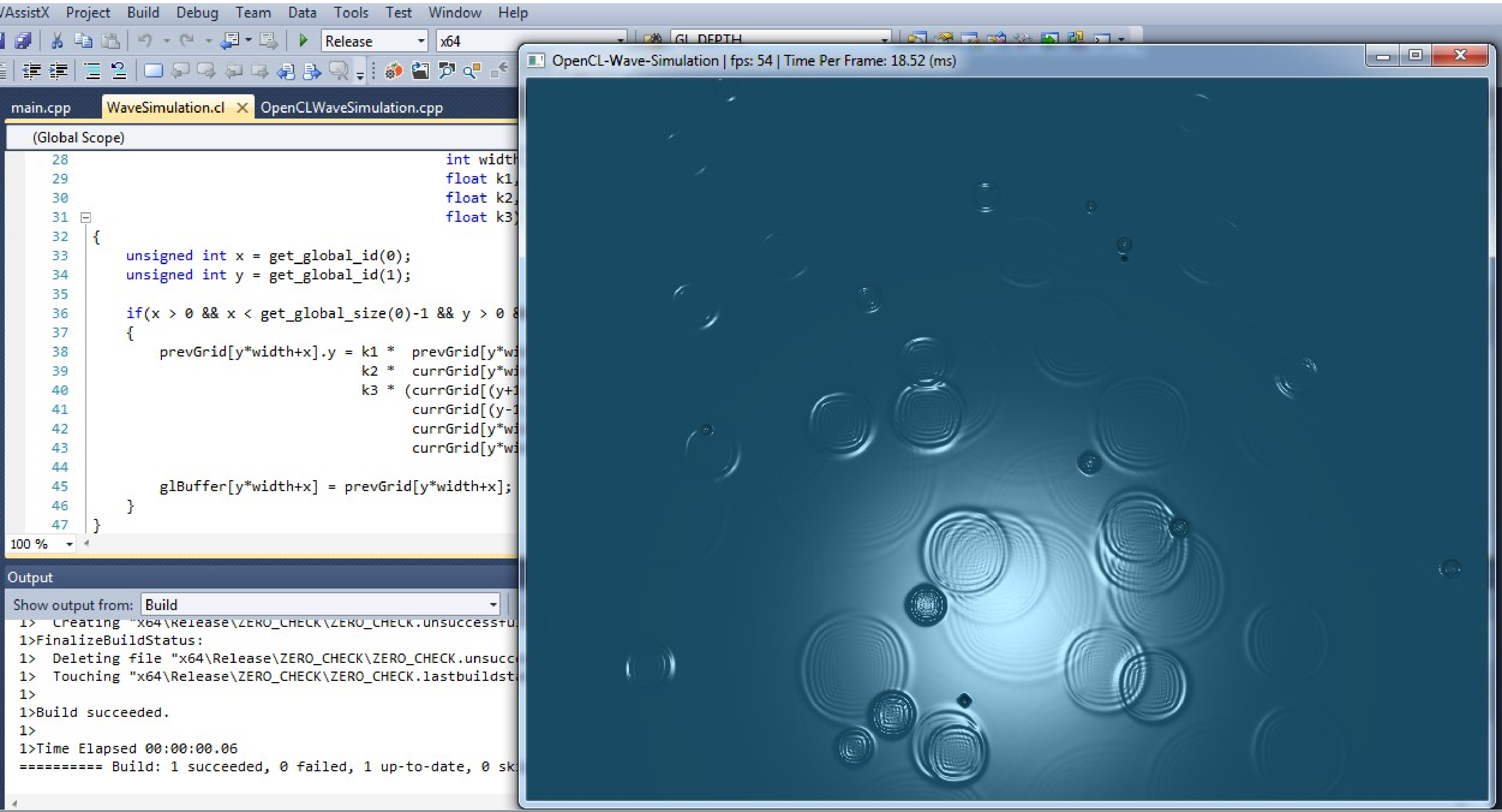
Software Profiling Course

Hannes Würfel

OUTLINE

1. Motivation
2. GPU Recap
3. OpenCL
4. CodeXL Overview
5. CodeXL Internals
6. CodeXL Profiling
7. CodeXL Debugging
8. Sources

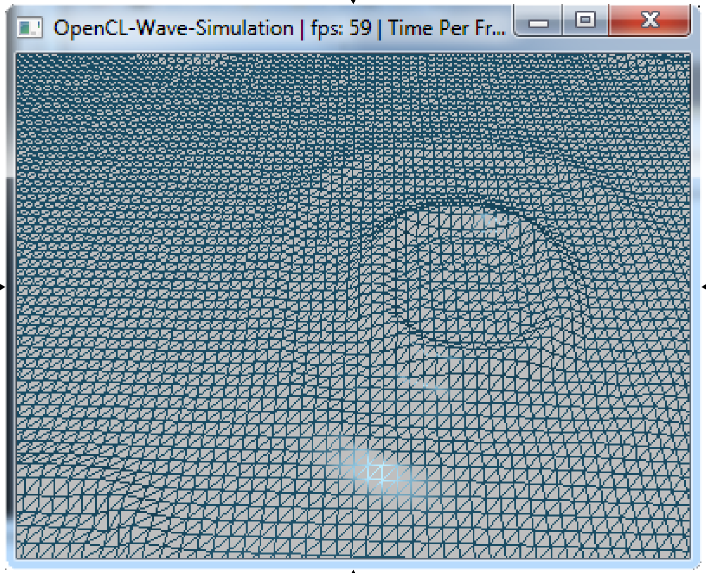
1. MOTIVATION



1. MOTIVATION

Vertex Displacement Kernel

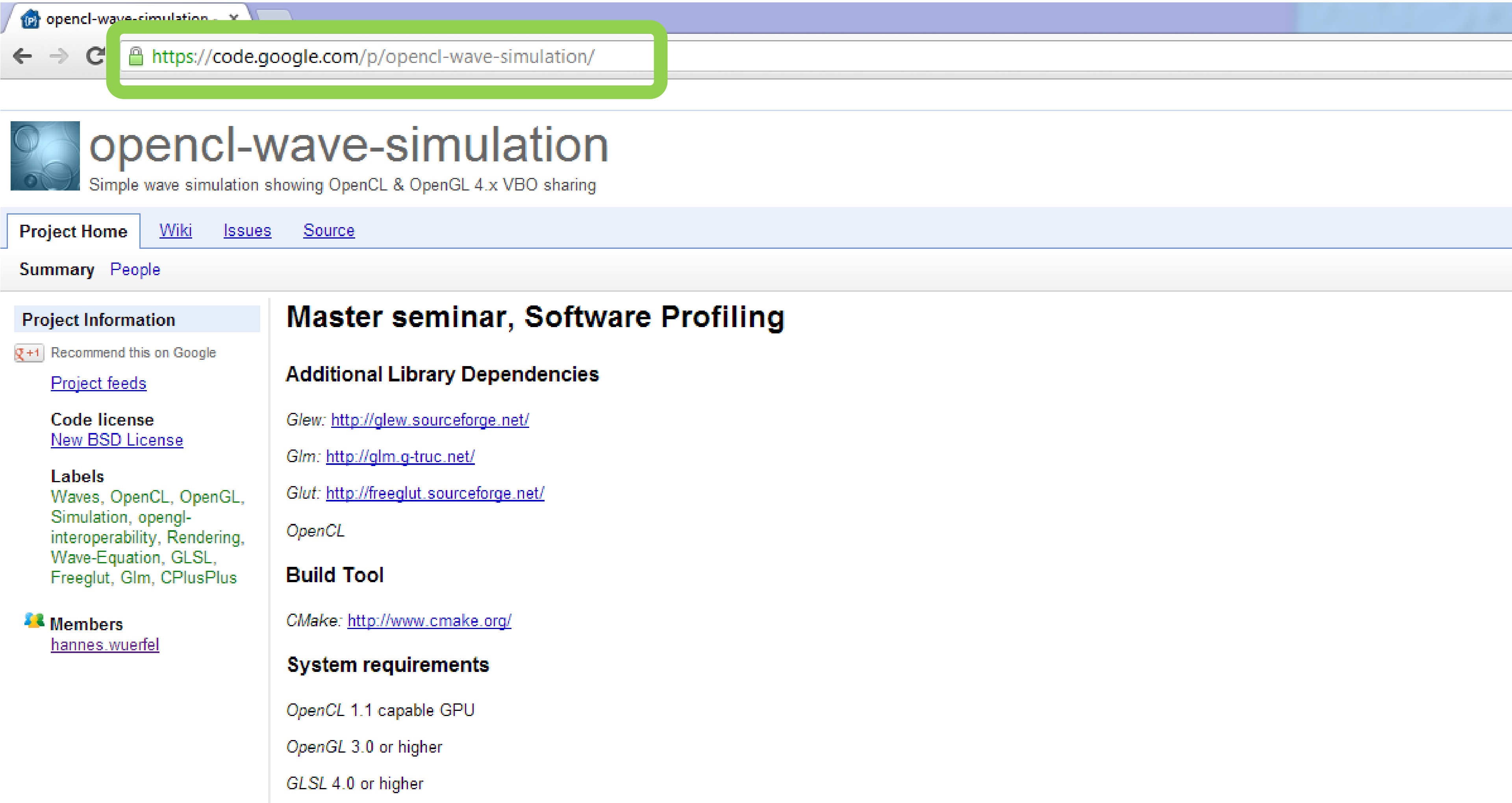
Initialize GL-Buffer Kernel



Disturb Grid Kernel

Finite Difference Scheme Kernel

1. MOTIVATION


opencl-wave-simulation

https://code.google.com/p/opencl-wave-simulation/

opencl-wave-simulation

Simple wave simulation showing OpenCL & OpenGL 4.x VBO sharing

Project Home | Wiki | Issues | Source

Summary | People

Project Information

Recommend this on Google

Project feeds

Code license
New BSD License

Labels
Waves, OpenCL, OpenGL, Simulation, opengl-interoperability, Rendering, Wave-Equation, GLSL, Freeglut, Glm, CPlusPlus

Members
hannes.wuerfel

Master seminar, Software Profiling

Additional Library Dependencies

Glew: http://glew.sourceforge.net/

Glm: http://glm.g-truc.net/

Glut: http://freeglut.sourceforge.net/

OpenCL

Build Tool

CMake: http://www.cmake.org/

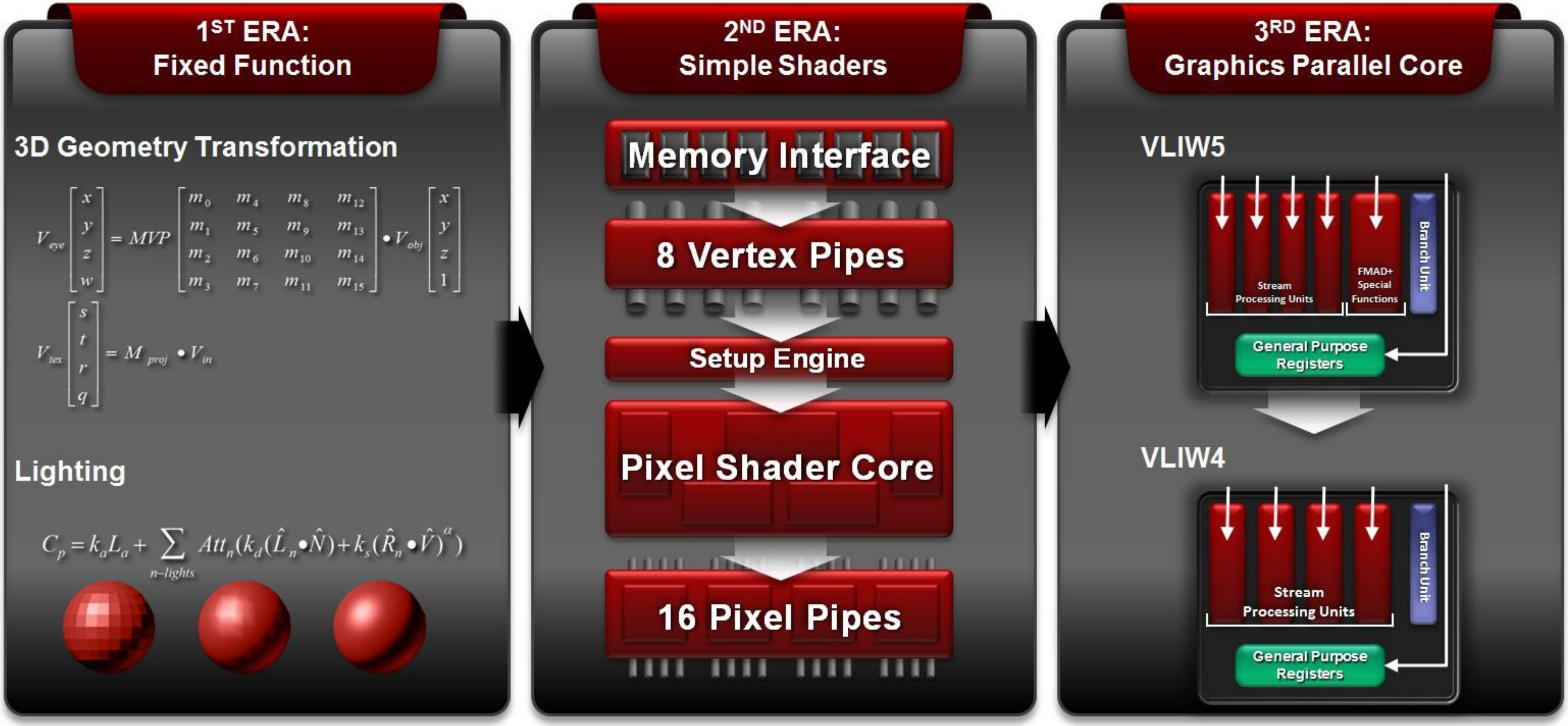
System requirements

OpenCL 1.1 capable GPU

OpenGL 3.0 or higher

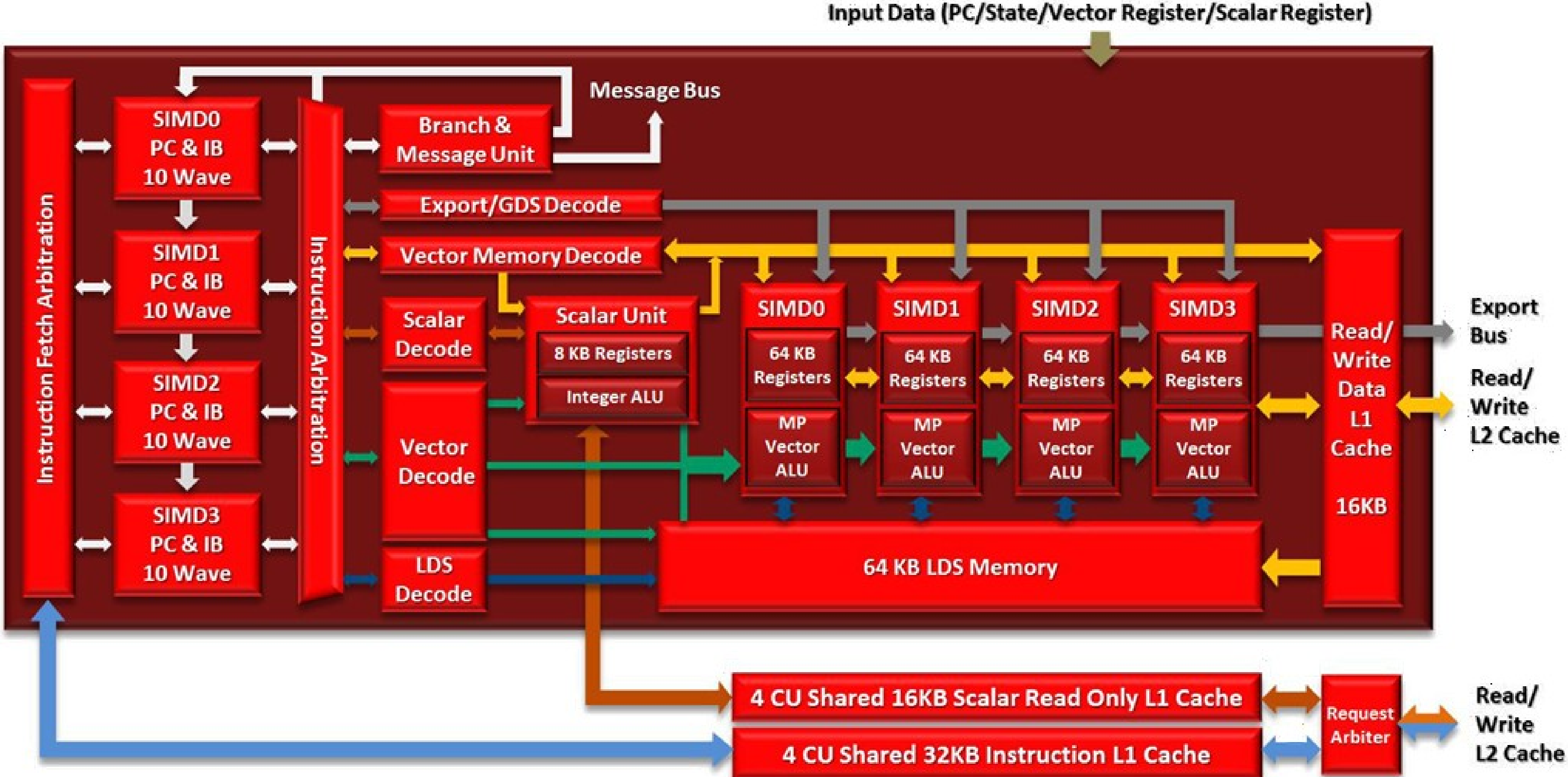
GLSL 4.0 or higher

2. GPU RECAP



http://www.amd.com/la/Documents/GCN_Architecture_whitepaper.pdf

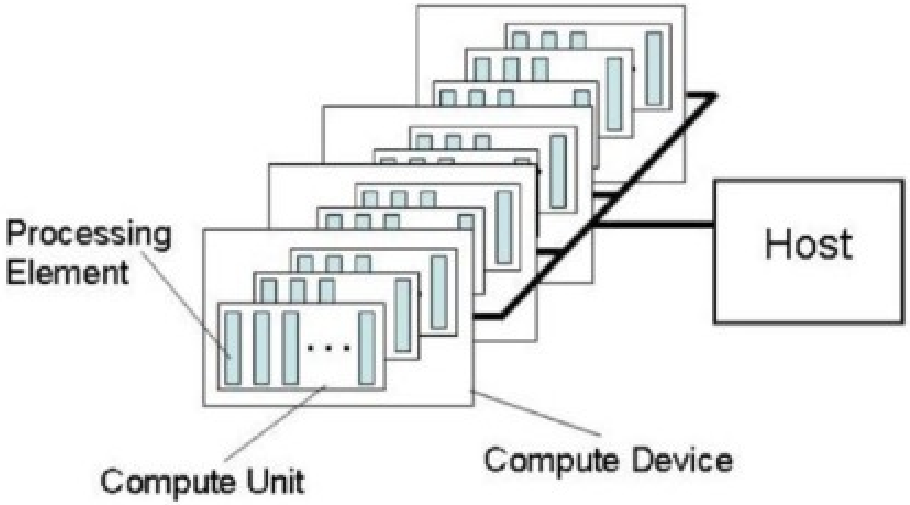
2. GPU RECAP



http://www.amd.com/la/Documents/GCN_Architecture_whitepaper.pdf

3. OPENCL

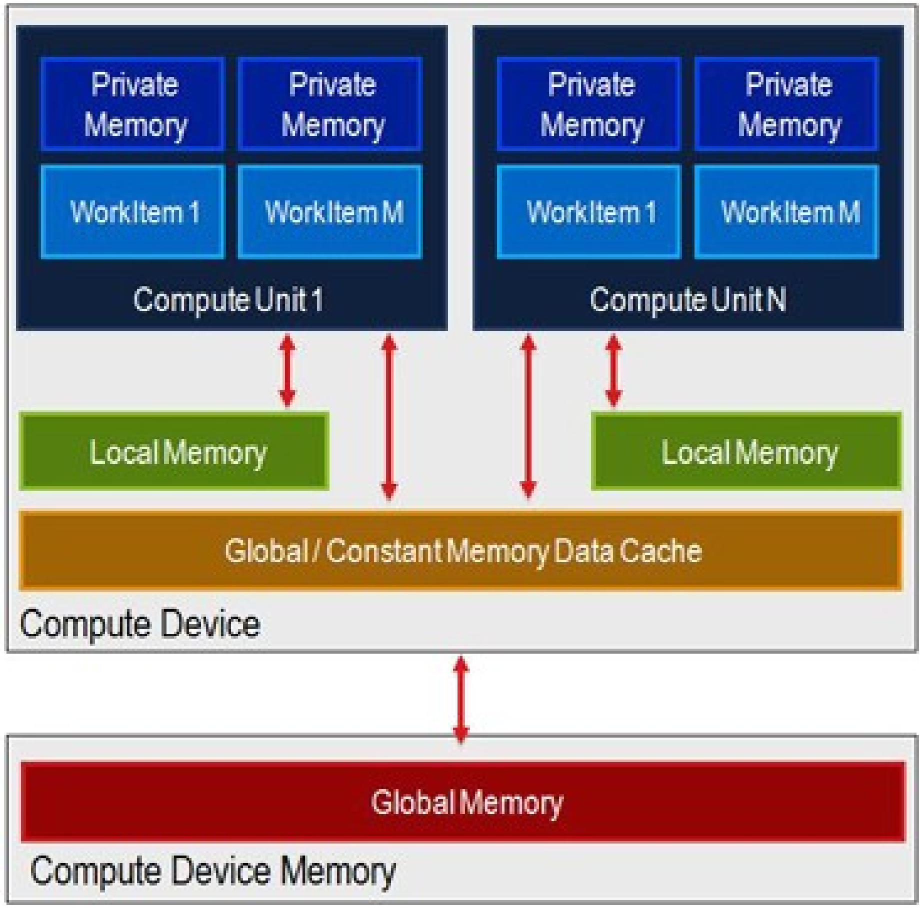
- Platform Model:



http://rastergrid.com/blog/2010/11/texture-and-buffer-access-performance/

3. OPENCL

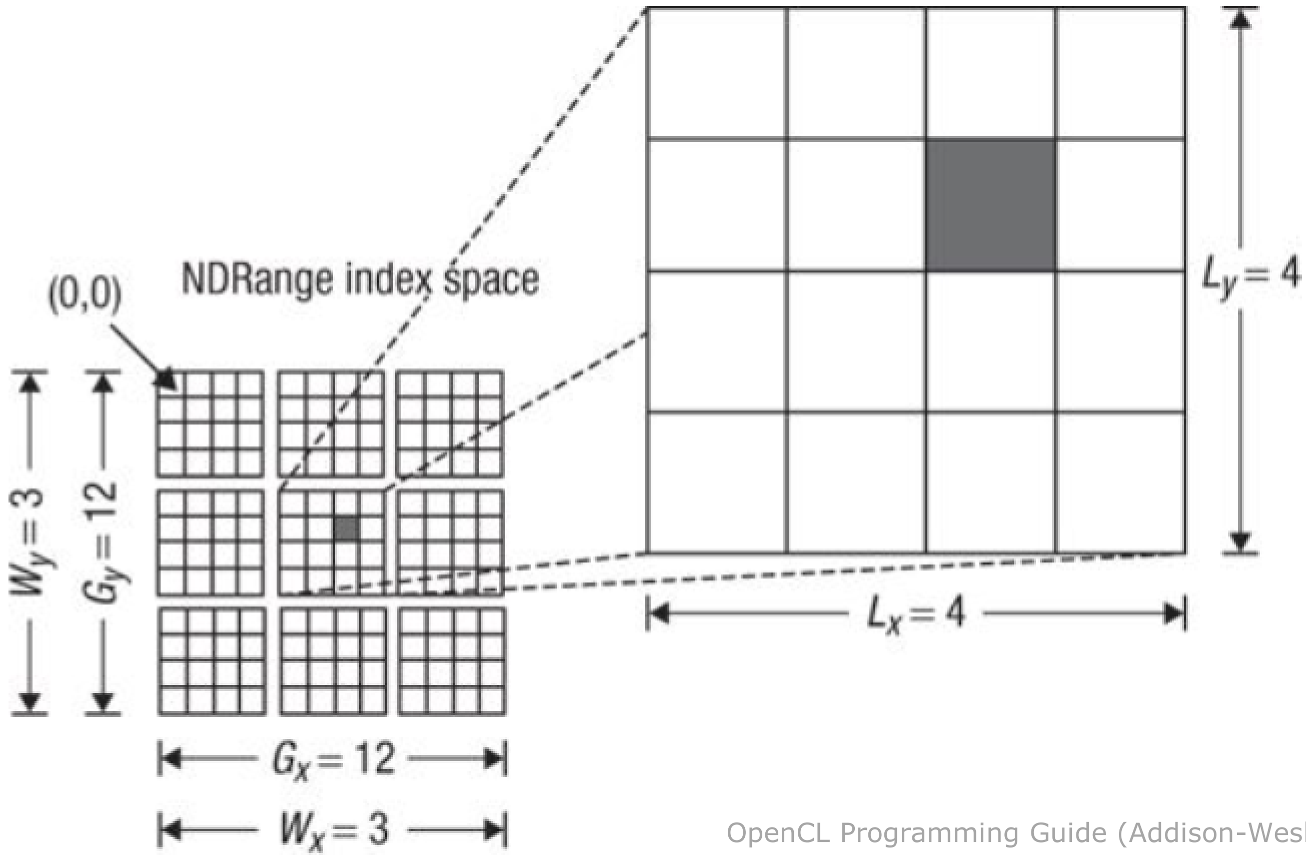
- Memory Hierarchy:



http://www.codeproject.com/Articles/122405/Part-2-OpenCL-Memory-Spaces

3. OPENCL

- Kernel Execution Model:



OpenCL Programming Guide (Addison-Wesley)

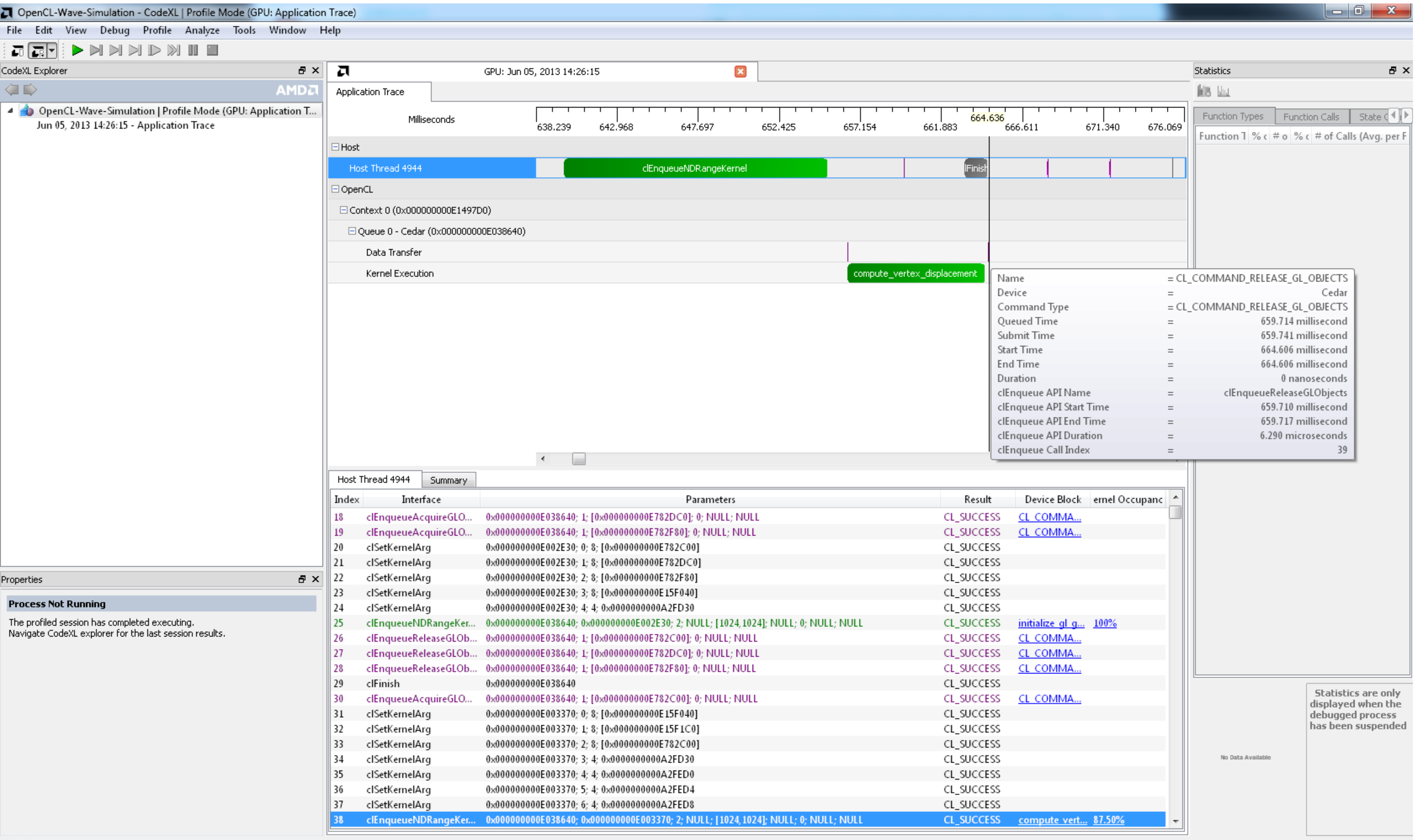
4. CODEXL OVERVIEW

- AMDs unified tool suite for profiling and debugging AMD CPUs, GPUs and APUs
- Former programs were:
  - gDebugger
  - APP Profiler
  - APP Kernel Analyzer
- Supported platforms:
  - Windows 7/8 (32-64Bit)
  - Red Hat Enterprise Linux 64Bit
  - Ubuntu 64Bit 12.04 or later
- Standalone application or Visual Studio 2010/2012 plugin

4. CODEXL OVERVIEW

- CPU Profiler
  - CPU Sampling
  - Call-Graph Profiling Features
- GPU Profiling
  - Application Trace
  - Hardware Performance Counters
  - Kernel Occupancy
  - Hotspots Analysis
- GPU Debugging
  - OpenGL & OpenCL API calls
  - OpenCL Kernel Debugging
  - DirectCompute Debugging
- Static Kernel Analysis
  - Hardware Disassembly Kernel Code

4. CODEXL OVERVIEW

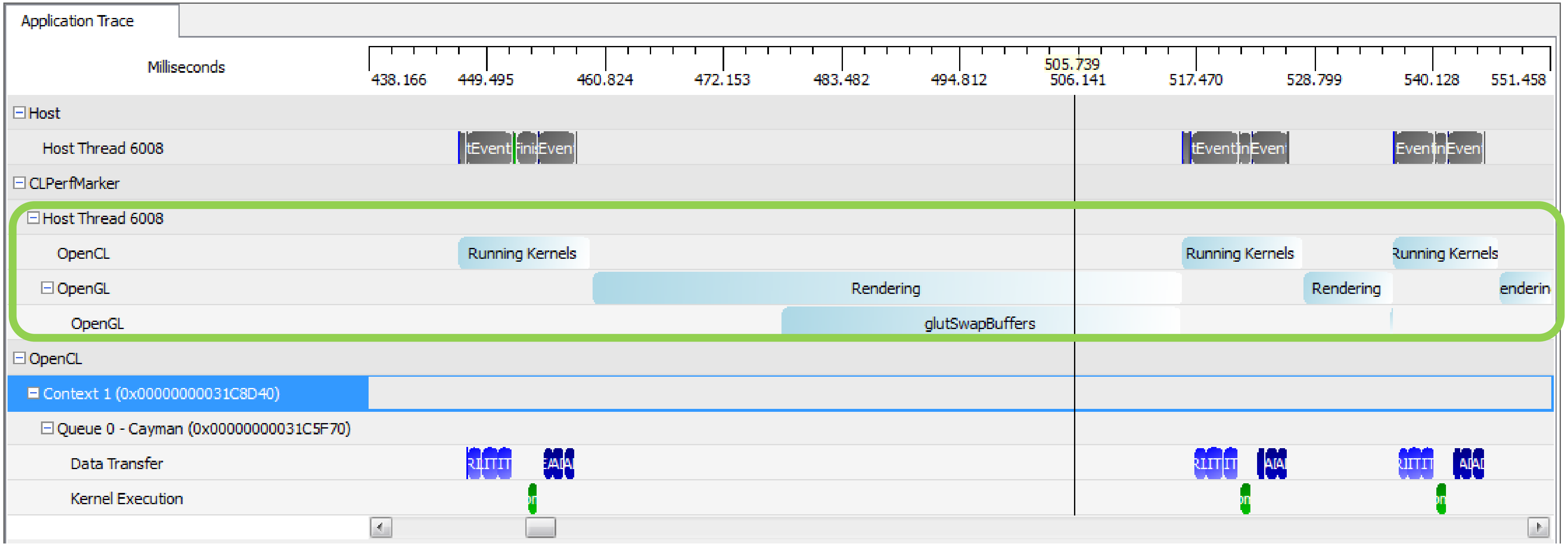


5. CODEXL INTERNALS

How does CodeXL Profiling works under the hood?

5. CODEXL INTERNALS

- Developers <u>can</u> instrument their source code by using the CLPerfMarkerAMD Library
  - *clBeginPerfMarkerAMD(), clEndPerfMarkerAMD()*



CodeXLHelp.chm

5. CODEXL INTERNALS

- Little information available

- Gathers data from OpenCL API run-time
- Uses GPU Perf API (AMD)
  - Provides derived counters based on raw Hardware performance counters
    - *Wavefronts, ALUStalledByLDS, ALUUtilization, …*
  - API uses a Sampling approach
    - ```
      GPA_Status GPA_BeginSample( gpa_uint32 sampleID );

      GPA_Status GPA_EndSample();
      ```
  - Needs Handle to current graphic context (OpenGL context/DirectX context) or Handle to an OpenCL command queue
      ```

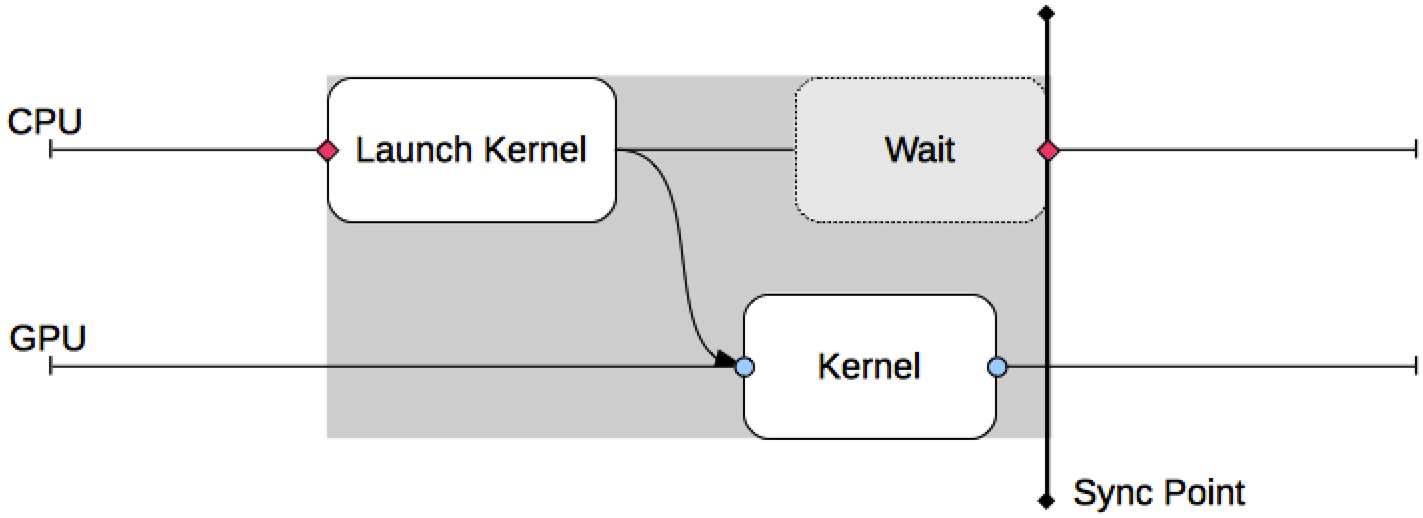
5. CODEXL INTERNALS

- Static/Dynamic binary instrumentation for HW performance counters and OpenCL API run-time?

- Educated guess: **Not at the application level, but ...**
  - Instrumentation at the GPU driver library level
  - Drivers provide callbacks for routines and capture measurements
  - Possible Methods:
    - Synchronous method
    - Event queue method
    - Callback method

5. CODEXL INTERNALS

- Synchronous Method:

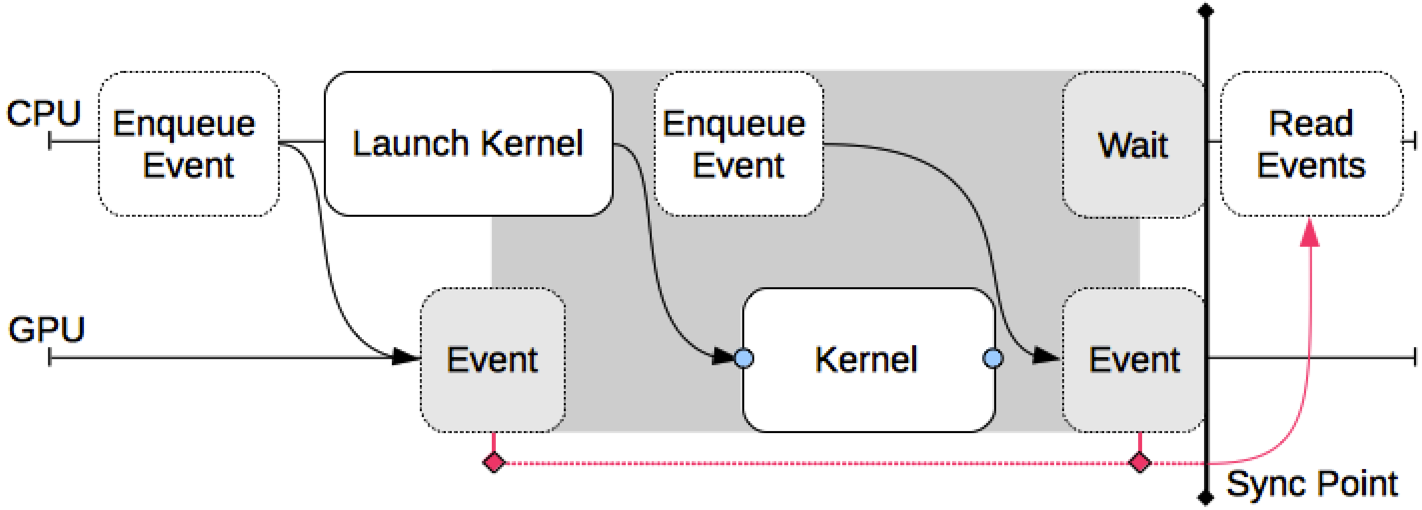


- Instrumentation around GPU API calls
- Implementation: wrap (synchronous) library with performance tool

Modified slides from TAU GPU Performance Measurement Tutorial

5. CODEXL INTERNALS

- Event queue method:

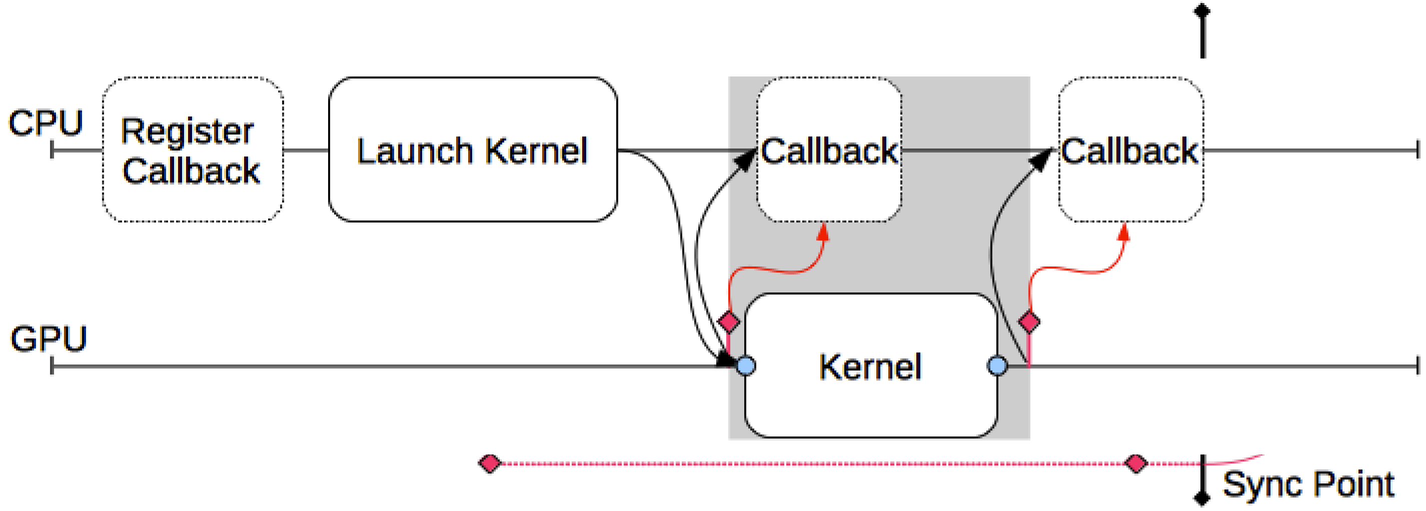


- Utilize OpenCL event support `clGetEventProfilingInfo`
- Instrumentation to create and insert events
- Implementation: driver library wrapping

Modified slides from TAU GPU Performance Measurement Tutorial

5. CODEXL INTERNALS

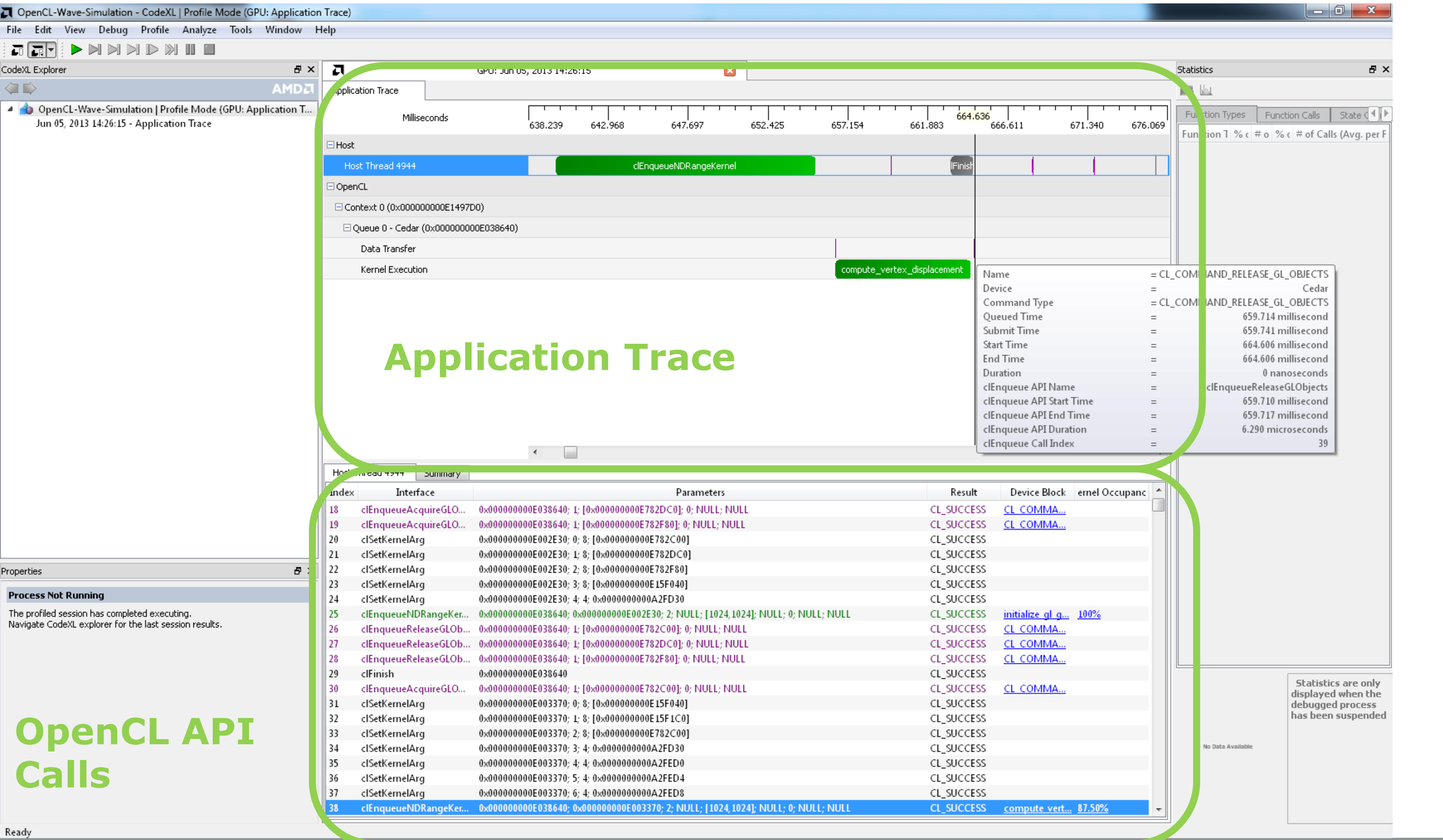
- Callback method:



- Utilize language-level callback support `clSetEventCallback`
- Implementation: Instrumentation to register callbacks

Modified slides from TAU GPU Performance Measurement Tutorial

5. CODEXL PROFILING



6. CODEXL PROFILING

- Summary Pages:

Host Thread 1628 | Summary

Previous | Next | Warning(s)/Error(s)

205	5654	1628	Best Practices	clEnqueueNDRangeKernel: Global Work size is too small - [1,1], resulting in low GPU utilization.
206	5683	1628	Best Practices	clEnqueueNDRangeKernel: Global Work size is too small - [1,1], resulting in low GPU utilization.
207	5712	1628	Best Practices	clEnqueueNDRangeKernel: Global Work size is too small - [1,1], resulting in low GPU utilization.
208	5741	1628	Best Practices	clEnqueueNDRangeKernel: Global Work size is too small - [1,1], resulting in low GPU utilization.
209	5770	1628	Best Practices	clEnqueueNDRangeKernel: Global Work size is too small - [1,1], resulting in low GPU utilization.
210	5799	1628	Best Practices	clEnqueueNDRangeKernel: Global Work size is too small - [1,1], resulting in low GPU utilization.
211	5828	1628	Best Practices	clEnqueueNDRangeKernel: Global Work size is too small - [1,1], resulting in low GPU utilization.
212	58	1628	Best Practices	Redundant synchronization detected. Synchronization API = clFinish
213	87	1628	Best Practices	Redundant synchronization detected. Synchronization API = clFinish

6. CODEXL PROFILING

- Summary Pages:

Host Thread 5572 | Summary

Previous | Next | API Summary

API Name	Cumulative Time(ms)	% of Total Time	# of Calls	Avg Time(ms)	Max Time(ms)	Min Time(ms)
clBuildProgram	422.62334	55.98005	1	422.62334	422.62334	422.62334
clFinish	149.77501	19.83897	210	0.71321	5.86820	0.06688
clCreateContext	47.79778	6.33122	1	47.79778	47.79778	47.79778
clCreateCommandQueue	45.03633	5.96544	1	45.03633	45.03633	45.03633
clEnqueueNDRangeKernel	35.66841	4.72458	312	0.11432	15.54999	0.00728
clEnqueueAcquireGLObjects	34.25837	4.53781	315	0.10876	1.92384	0.05463
clCreateBuffer	15.94595	2.11218	2	7.97297	7.99830	7.94765
clCreateFromGLBuffer	2.46779	0.32688	3	0.82260	1.95033	0.18176
clEnqueueReleaseGLObjects	1.03361	0.13691	315	0.00328	0.01059	0.00199
clSetKernelArg	0.32179	0.04262	1768	0.00018	0.00761	0.00033
clCreateProgramWithSource	0.01258	0.00167	1	0.01258	0.01258	0.01258
clCreateKernel	0.00596	0.00079	4	0.00149	0.00298	0.00099
clGetDeviceIDs	0.00463	0.00061	1	0.00463	0.00463	0.00463

6. CODEXL PROFILING

- Summary Pages:

Host Thread 1628 | Summary

Previous | Next | Kernel Summary

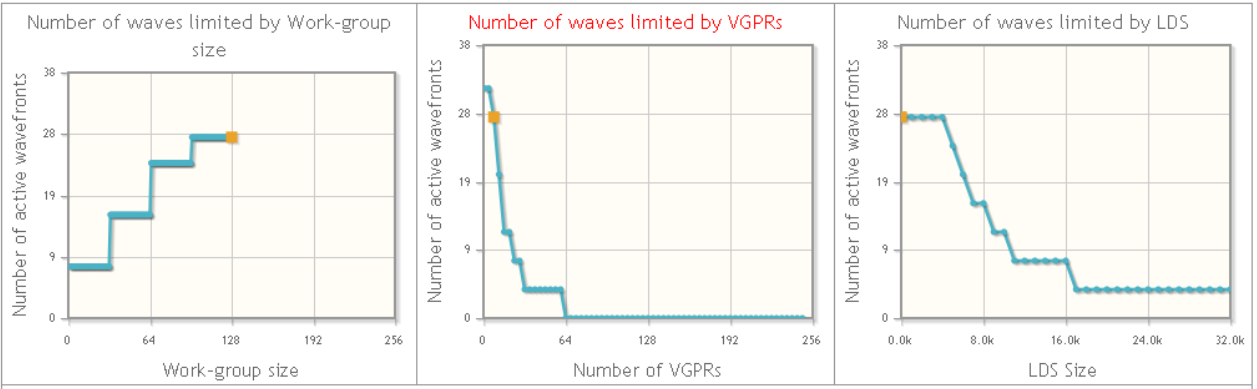
Kernel Name	Device Name	# of Calls	Total Time(ms)	Avg Time(ms)	Max Time(ms)	Min Time(ms)
compute_vertex_displacement	Cedar	212	1698.98523	8.01408	8.03633	7.99089
compute_finite_difference_scheme	Cedar	212	1179.43522	5.56337	6.12511	5.52711
initialize_gl_grid	Cedar	1	6.47633	6.47633	6.47633	6.47633
disturb_grid	Cedar	211	2.57678	0.01221	0.01578	0.00989

6. CODEXL PROFILING

Hasso Plattner Institut

- Summary Pages:
  - Context Summary Page
  - Top 10 Data Transfer Summary Page
  - Top 10 Kernel Summary Page

6. CODEXL PROFILING



Variable	Value	Device Limit
Device Info		
Device name	Cedar	
Number of compute units	2	
Max number of waves per compute unit	32	
Max number of work-groups per compute unit	8	
Wavefront size	32	
Kernel Info		
Kernel name	compute_vertex_displacement	
Vector GPR usage per work-item	8	248
LDS usage per work-group	0	32768
Flattened work-group size	128	128
Flattened global work size	1048576	2097152
Number of waves per work-group	4	4
Kernel Occupancy		
Number of waves limited by Vector GPR and Work-group size	28	32
Number of waves limited by LDS and Work-group size	32	32
Number of waves limited by Work-group size	32	32
Limiting factor(s)	**VGPR**	
Estimated occupancy	87.5%	

- Shows utilization of a Compute Unit
- Measured by number of in-flight wavefronts for a given Kernel, relative to the maximum number of wavefronts given an ideal Kernel dispatch configuration

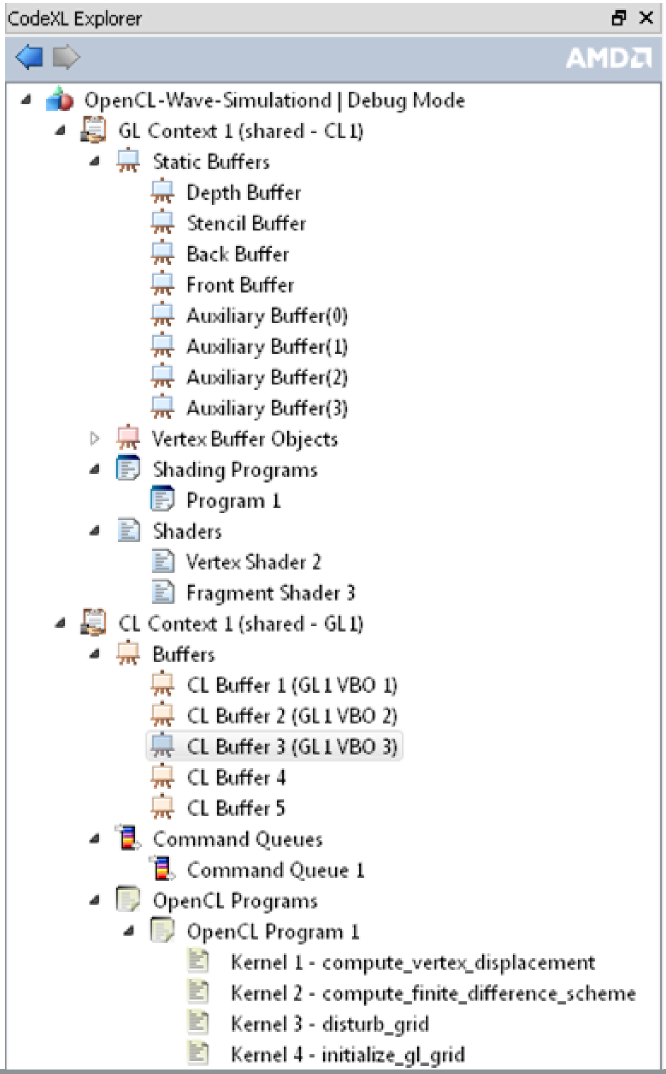
6. CODEXL PROFILING

- HW Performance Counters:

Method	ExecutionOrder	ThreadID	CallIndex	GlobalWorkSize	WorkGroupSize	Time	VGPRs	SGPRs	FCStacks	KernelOccupancy	Wavefronts	ALUInsts	FetchInsts	WriteInst
initialize_gl_grid__k1_Cedar1	1	5316	25	{ 1024 1024 1}	NULL	648456	4	NA	0	100	32768	9	1	3
compute_vertex_displacement__k2_Cedar1	2	5316	38	{ 1024 1024 1}	NULL	805278	8	NA	1	87.5	32768	23.97	5.99	2
compute_finite_difference_scheme__k3_Cedar1	3	5316	48	{ 1024 1024 1}	NULL	553656	7	NA	1	100	32768	33.95	3.99	2
disturb_grid__k4_Cedar1	4	5316	57	{ 1 1 1}	NULL	001311	5	NA	0	100	1	11	5	5
compute_vertex_displacement__k2_Cedar1	5	5316	66	{ 1024 1024 1}	NULL	800544	8	NA	1	87.5	32768	23.97	5.99	2
compute_finite_difference_scheme__k3_Cedar1	6	5316	76	{ 1024 1024 1}	NULL	557278	7	NA	1	100	32768	33.95	3.99	2
disturb_grid__k4_Cedar1	7	5316	85	{ 1 1 1}	NULL	.01000	5	NA	0	100	1	11	5	5

ALUBusy (%)	ALUFetchRatio	ALUPacking (%)	FetchSize	CacheHit (%)	FetchUnitBusy (%)	FetchUnitStalled (%)	WriteUnitStalled (%)
14.06		55.56	16384	0	3.13	0	62.93
30.13		43.32	50277.75	24.17	29.51	15.29	79.51
61.18	.51	43.52	24201.25	51.55	14.64	0.29	71.16
0.32	.20	58.18	0.31	0	0.50	0	0
30.05		43.32	50275.75	24.17	28.65	14.25	80
61.53	.51	43.52	24226.06	51.50	14.74	0.30	70.92
0.33	2.20	58.18	0.31	0	0.50	0	0

7. CODEXL DEBUGGING

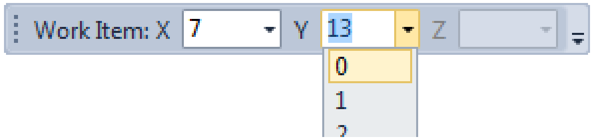


- OpenCL and OpenGL objects
- Shared contexts
- Shader and Kernel resources
- Ability to show buffer contents

7. CODEXL DEBUGGING

Hasso Plattner Institut

- Kernel code breakpoints
- Stepping through one Kernel instance
- Switching between Kernel instances



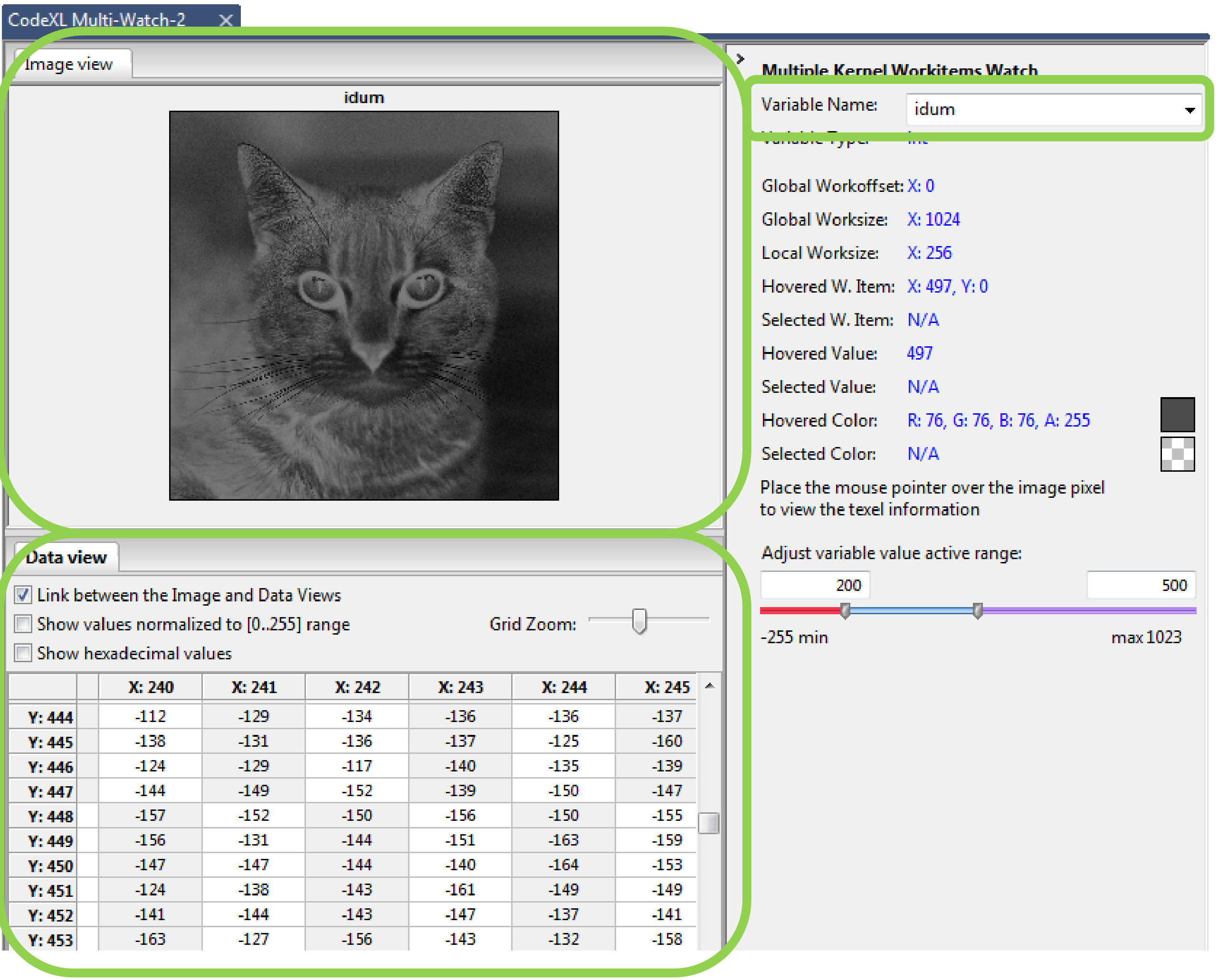

```
__kernel void compute_finite_difference_scheme(__global float4* currGrid,
                                               __global float4* glNormalBuffer,
                                               __global float4* glTangentBuffer,
                                               int width,
                                               float spatialStep)
{
    unsigned int x = get_global_id(0);
    unsigned int y = get_global_id(1);

    if(x > 0 && x < get_global_size(0)-1 && y > 0 && y < get_global_size(1)-1)
    {
        float l = currGrid[y*width+(x-1)].y;
        float r = currGrid[y*width+(x+1)].y;
        float t = currGrid[(y-1)*width+x].y;
        float b = currGrid[(y+1)*width+x].y;

        float4 estimatedNormal  = (float4)(l-r, 2.0f*spatialStep, b-t, 1.0f);
        float4 estimatedTangent = (float4)(2.0f*spatialStep, r-l, 0.0f, 1.0f);

        glNormalBuffer[y*width+x]  = normalize(estimatedNormal);
        glTangentBuffer[y*width+x] = normalize(estimatedTangent);
    }
}
```

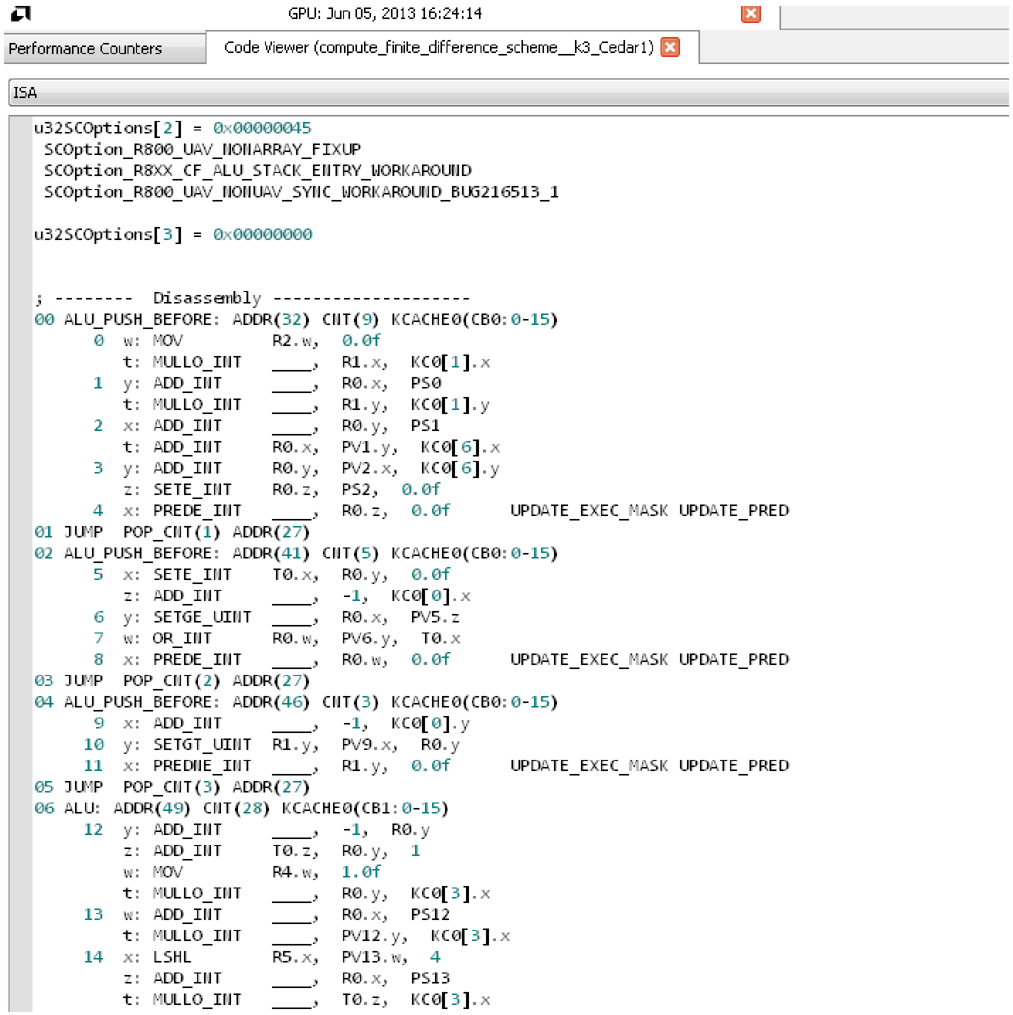
7. CODEXL DEBUGGING



- Multi-Watch View
- Choose variable to inspect
- Variable across all work items
- Visualization of the buffer

CodeXLHelp.chm

7. OPENCL DEBUGGING




```
u32SCOptions[2] = 0x00000045
 SCOption_R800_UAV_NONARRAY_FIXUP
 SCOption_R8XX_CF_ALU_STACK_ENTRY_WORKAROUND
 SCOption_R800_UAV_NONUAV_SYNC_WORKAROUND_BUG216513_1

u32SCOptions[3] = 0x00000000


; --------  Disassembly --------------------
00 ALU_PUSH_BEFORE: ADDR(32) CNT(9) KCACHE0(CB0:0-15)
      0  w: MOV         R2.w,   0.0f
         t: MULLO_INT   ____,   R1.x,   KC0[1].x
      1  y: ADD_INT     ____,   R0.x,   PS0
         t: MULLO_INT   ____,   R1.y,   KC0[1].y
      2  x: ADD_INT     ____,   R0.y,   PS1
         t: ADD_INT     R0.x,   PV1.y,   KC0[6].x
      3  y: ADD_INT     R0.y,   PV2.x,   KC0[6].y
         z: SETE_INT    R0.z,   PS2,   0.0f
      4  x: PREDE_INT   ____,   R0.z,   0.0f      UPDATE_EXEC_MASK UPDATE_PRED
01 JUMP  POP_CNT(1) ADDR(27)
02 ALU_PUSH_BEFORE: ADDR(41) CNT(5) KCACHE0(CB0:0-15)
      5  x: SETE_INT    T0.x,   R0.y,   0.0f
         z: ADD_INT     ____,   -1,   KC0[0].x
      6  y: SETGE_UINT  ____,   R0.x,   PV5.z
      7  w: OR_INT      R0.w,   PV6.y,   T0.x
      8  x: PREDE_INT   ____,   R0.w,   0.0f      UPDATE_EXEC_MASK UPDATE_PRED
03 JUMP  POP_CNT(2) ADDR(27)
04 ALU_PUSH_BEFORE: ADDR(46) CNT(3) KCACHE0(CB0:0-15)
      9  x: ADD_INT     ____,   -1,   KC0[0].y
     10  y: SETGT_UINT  R1.y,   PV9.x,   R0.y
     11  x: PREDNE_INT  ____,   R1.y,   0.0f      UPDATE_EXEC_MASK UPDATE_PRED
05 JUMP  POP_CNT(3) ADDR(27)
06 ALU: ADDR(49) CNT(28) KCACHE0(CB1:0-15)
     12  y: ADD_INT     ____,   -1,   R0.y
         z: ADD_INT     T0.z,   R0.y,   1
         w: MOV         R4.w,   1.0f
         t: MULLO_INT   ____,   R0.y,   KC0[3].x
     13  w: ADD_INT     ____,   R0.x,   PS12
         t: MULLO_INT   ____,   PV12.y,   KC0[3].x
     14  x: LSHL        R5.x,   PV13.w,   4
         z: ADD_INT     ____,   R0.x,   PS13
         t: MULLO_INT   ____,   T0.z,   KC0[3].x
```

- Static Kernel analyzer
- Allows to compile, to analyze and to disassemble OpenCL Kernel code for multiple device versions
- (also DirectCompute Kernels)

SUBJECTIVE EVALUATION

- ✅ Application trace provides useful information about concurrent activities in the program
- ✅ Best Practices as unnecessary API calls, …
- ✅ Kernel debugging
- ✅ Multi-View to detect errors in bound checks, …

- ❌ Stepping through a Kernel took too long on my test system
- ❌ Lack of insights in documentation

8. SOURCES

- OpenCL Programming Guide (Addison Wesley 2012)
- CodeXL User Guide
- Mathematics for 3D Game Programming and Computer Graphics (Course Technology PTR 3rd Edition 2012)
- http://developer.amd.com/tools-and-sdks/heterogeneous-computing/codexl/
- http://developer.amd.com/tools-and-sdks/graphics-development/gpuperfapi/
- http://www.amd.com/la/Documents/GCN_Architecture_whitepaper.pdf
- http://www.cc.gatech.edu/~vetter/keeneland/tutorial-2011-04-14/10-tau-gpu-tutorial-part1.pdf
- http://www.nvidia.com/content/nvision2008/tech_presentations/Professional_Visualization/NVISION08-Advanced_OpenGL_Debugger.pdf