Advanced CUDA Programming

CUDA Development Tools

GPU Tools

- Profiler
  - Available for all supported OSs
  - Command-line or GUI
  - Sampling signals on GPU for:
    - Memory access parameters
    - Execution (serialization, divergence)
- Debugger
  - Windows: Parallel Nsight, Linux: cuda-gdb
  - Debug directly on the GPU

cuda-gdb: CUDA Application Debugging

- *Included with CUDA toolkit*
  - *Superset of GDB commands to support GPU programming*
  - *Specific "cuda" commands to navigate threads*
  - *Support for instruction level debugging within a CUDA kernel*
- *Compile with "`nvcc -g -G`" flags for symbols*
- *Invoke on command line or as backend to DDD, Emacs, etc…*
- *Documentation: `/usr/local/cuda/doc/CUDA_GDB_v3-0.pdfs`*

File Edit Options Buffers Tools Gud Help

```
        } else {
            acos_noftz_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams);
        }
    } else {
        if (opts.ieee == 3) {
            acos_ieee3_ftz_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams);
        } else if (opts.ieee == 2) {
            acos_ieee2_ftz_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams);
        } else if (opts.ieee == 1) {
            acos_ieee1_ftz_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams);
        } else {
            acos_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams
        }
    }
#else /* FERMI */
    acos_main<<<ACOS_CTA_CNT,ACOS_THREAD_CNT>>>(funcParams);
#endif
    stop = second();
    cudaStat = cudaGetLastError(); /* check for launch error */

    if (cudaStat != cudaSuccess) {
        fprintf (stderr, "!!!! program launch failed\n");
        CLEANUP();
        return EXIT_FAILURE;
    }
    fprintf (stdout, "^^^^ elapsed = %10.8f sec  Gfuncs/sec=%g\n",
             (stop-start), (1e-9*funcParams.n)/(stop-start));

    cudaStat = cudaMemcpy (res, acosRes, opts.n * sizeof(res[0]),
                           cudaMemcpyDeviceToHost);
    if (cudaStat != cudaSuccess) {
```

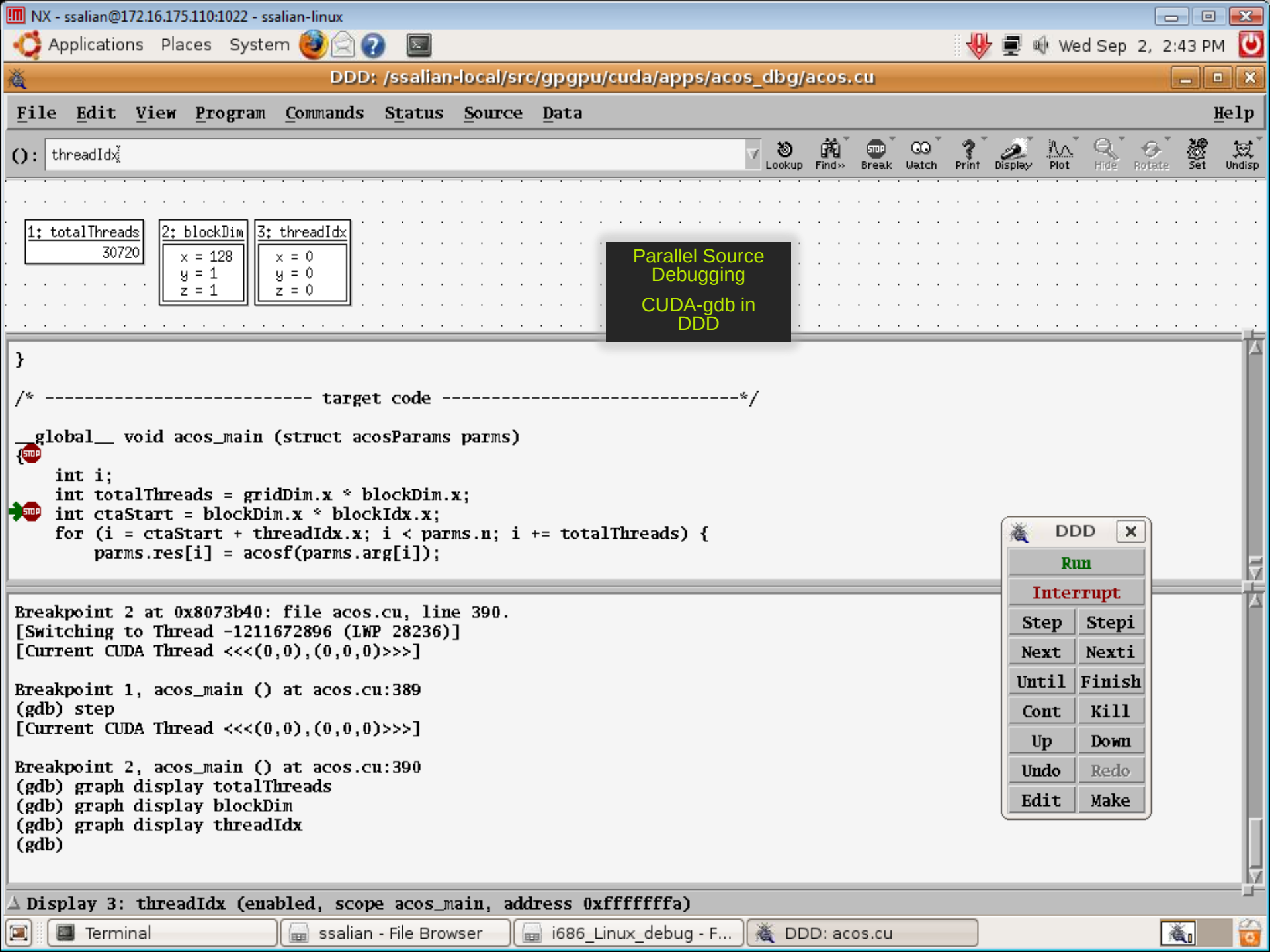
-1:%% math_functions.h 43% L2192 (C/l Abbrev)----------

Parallel Source Debugging

CUDA-gdb in emacs

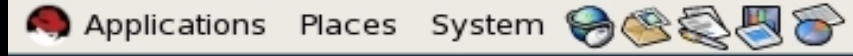
```
__device_func__(float __cuda_acosf(float a))
{
  float t0, t1, t2;

  t0 = __cuda_fabsf(a);
  t2 = 1.0f - t0;
  t2 = 0.5f * t2;
  t2 = __cuda_sqrtf(t2);
  t1 = t0 > 0.57f ? t2 : t0;
  t1 = __internal_asinf_kernel(t1);
  t1 = t0 > 0.57f ? 2.0f * t1 : CUDART_PIO2_F - t1;
  if (__cuda___signbitf(a)) {
    t1 = CUDART_PI_F - t1;
  }
#if !defined(__CUDABE__)
  if (__cuda___isnanf(a)) {
    t1 = a + a;
  }
#endif
  return t1;
```

```
Breakpoint 1, acos_main () at acos.cu:389
(cuda-gdb) s
[Current CUDA Thread <<<(0,0),(0,0,0)>>>]
acos_main () at acos.cu:390
(cuda-gdb) info cuda lane
DEV:  0/1    Device Type: gt200   SM Type: sm_13   SM/WP/LN: 30/32/32   Regs/LN: 128nSM:  0/30   valid warps: 00
WP:  0/32   valid/active/divergent lanes: 0xffffffff/0xffffffff/0x00000000   block: (0,0)
LN:  0/32   pc=0x0000000000000050   thread: (0,0,0)
(cuda-gdb) info cuda threads
<<<(0,0),(0,0,0)>>> ... <<<(0,0),(31,0,0)>>> acos_main () at acos.cu:390
<<<(0,0),(32,0,0)>>> ... <<<(239,0),(127,0,0)>>> acos_main () at acos.cu:389
(cuda-gdb) s
```


NX - ssalian@172.16.175.110:1022 - ssalian-linux
Applications Places System
Wed Sep 2, 2:43 PM
DDD: /ssalian-local/src/gpgpu/cuda/apps/acos_dbg/acos.cu
File Edit View Program Commands Status Source Data Help
(): threadIdx
Lookup Find» Break Watch Print Display Plot Hide Rotate Set Undisp
1: totalThreads
30720
2: blockDim
x = 128
y = 1
z = 1
3: threadIdx
x = 0
y = 0
z = 0
Parallel Source Debugging
CUDA-gdb in DDD
}
/* ---------- target code ----------*/
__global__ void acos_main (struct acosParams parms)
{
int i;
int totalThreads = gridDim.x * blockDim.x;
int ctaStart = blockDim.x * blockIdx.x;
for (i = ctaStart + threadIdx.x; i < parms.n; i += totalThreads) {
parms.res[i] = acosf(parms.arg[i]);
Breakpoint 2 at 0x8073b40: file acos.cu, line 390.
[Switching to Thread -1211672896 (LWP 28236)]
[Current CUDA Thread <<<(0,0),(0,0,0)>>>]
Breakpoint 1, acos_main () at acos.cu:389
(gdb) step
[Current CUDA Thread <<<(0,0),(0,0,0)>>>]
Breakpoint 2, acos_main () at acos.cu:390
(gdb) graph display totalThreads
(gdb) graph display blockDim
(gdb) graph display threadIdx
(gdb)
DDD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Cont Kill
Up Down
Undo Redo
Edit Make
Display 3: threadIdx (enabled, scope acos_main, address 0xfffffffa)
Terminal
ssalian - File Browser
i686_Linux_debug - F...
DDD: acos.cu

CUDA-MemCheck

- Available with CUDA 3.0 Release
- Track out of bounds and misaligned accesses
- Supports CUDA C
- Integrated into the CUDA-GDB debugger
- Available as standalone tool on all OS platforms.



jchase@dhcp-172-16-175-68:/src/gpgpu/bin/i686_

File Edit View Terminal Tabs Help

```
[jchase@dhcp-172-16-175-68 i686_Linux_debug]$ cuda-memcheck ./ptrchecktest
========= CUDA-MEMCHECK
Checking...
Done
Checking...
Error: 3 (65538)
Done
Checking...
Error: 0 (1)
Error: 1 (0)
Error: 2 (0)
Error: 3 (0)
Error: 4 (0)
Error: 5 (0)
Error: 6 (0)
Error: 7 (0)
Done
unspecified launch failure : 125
========= Invalid read of size 4
=========     at 0x000000f0 in kernel2 (/src/gpgpu/cudamemcheck/test/ptrchecktest.cu:27)
=========     by thread 5 in block 3
========= Address 0x00101015 is misaligned
=========
========= Invalid read of size 4
=========     at 0x000000f0 in kernel1 (/src/gpgpu/cudamemcheck/test/ptrchecktest.cu:18)
=========     by thread 3 in block 5
========= Address 0x00101028 is out of bounds
=========
========= Invalid write of size 8
=========     at 0x00000170 in kernel3 (/src/gpgpu/cudamemcheck/test/ptrchecktest.cu:38)
=========     by thread 1 in block 8
========= Address 0x00102004 is misaligned
=========
========= Invalid write of size 4
=========     at 0x000000a0 in kernel4 (/src/gpgpu/cudamemcheck/test/ptrchecktest.cu:44)
=========     by thread 63 in block 22
========= Address 0x00000000 is out of bounds
=========
========= ERROR SUMMARY: 4 errors
[jchase@dhcp-172-16-175-68 i686_Linux_debug]$
```

Parallel Source Memory Checker

CUDA-MemCheck

cuda-gdb: Demo

- *Debugging* example code:
  `bitreverse_debugger.cu`

- Debugging example code:
  `bitreverse_debugger-infloop.cu`

cudaprof: CUDA application profiling

- Included as part of CUDA toolkit
  - Command line usage with no re-compile
  - Configurable through environment variables
  - Low overhead hardware counters
  - Measures both instruction and memory operations
- Set environmental variable: `CUDA_PROFILE=1`
  - Run application as normal
  - Examine profile output: `cuda_profile_0.log`
  - Configure options and four active profile signals via a configuration file: `CUDA_PROFILE_CONFIG=configuration-file`
- Visual profiler: `cudaprof` provides ease of use and enhanced reporting
- Documentation: `/usr/local/cuda/docs/CUDA_Profiler_3.0.txt`

Cudaprof Collection Options

The profiler supports the following options:

- timestamp : Time stamps for kernel launches and memory transfers. This can be used for timeline analysis.
- gpustarttimestamp : Time stamp when kernel starts execution in GPU.
- gpuendtimestamp : Time stamp when kernel ends execution in GPU.
- gridsize : Number of blocks in a grid along the X and Y dimensions for a kernel launch
- threadblocksize : Number of threads in a block along the X, Y and Z dimensions for a kernel launch
- dynsmemperblock : Size of dynamically allocated shared memory per block in bytes for a kernel launch
- stasmemperblock : Size of statically allocated shared memory per block in bytes for a kernel launch
- regperthread : Number of registers used per thread for a kernel launch.
- memtransferdir : Memory transfer direction, a direction value of 0 for host->device memory copies and a value of 1 for device->host memory copies.
- memtransfersize : Memory copy size in bytes
- memtransferhostmemtype : Host memory type (pageable or page-locked)
- streamid : Stream Id for a kernel launch

Cudaprof Collection Signals

The profiler supports logging of following counters during kernel execution on all architectures:

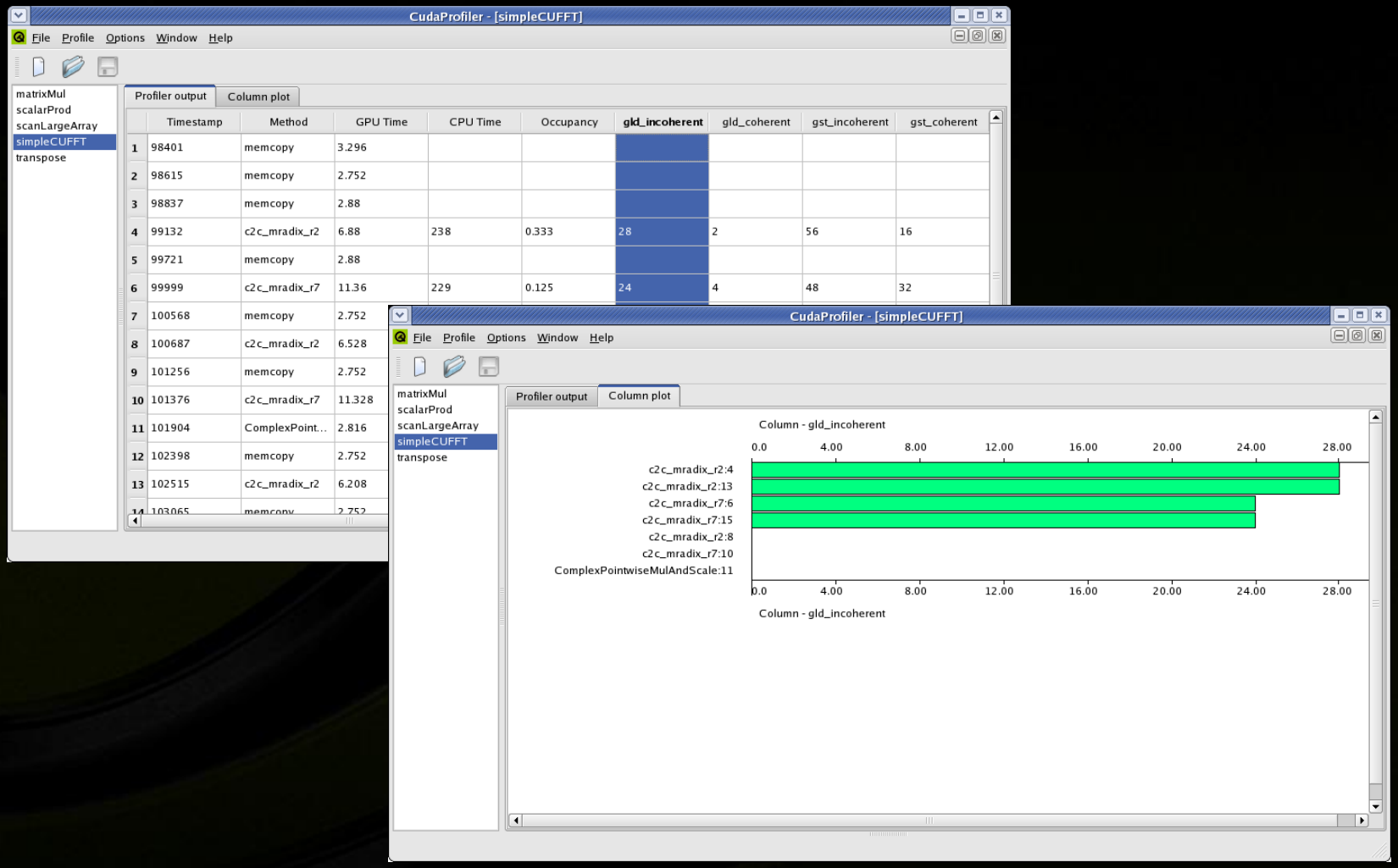
- local_load : Number of executed local load instructions per warp in a SM
- local store : Number of executed local store instructions per warp in a SM
- gld_request : Number of executed global load instructions per warp in a SM
- gst_request : Number of executed global store instructions per warp in a SM
- divergent_branch : Number of unique branches that diverge
- branch : Number of unique branch instructions in program
- sm_cta_launched : Number of threads blocks executed on a SM

Cudaprof Collection Signals

The profiler supports logging of following counters during kernel execution only on GPUs with Compute Capability 1.x:

- **gld_incoherent : Non-coalesced (incoherent) global memory loads**
- **gld_coherent : Coalesced (coherent) global memory loads**
- **gld_32b : 32-byte global memory load transactions**
- **gld_64b : 64-byte global memory load transactions**
- **gld_128b : 128-byte global memory load transactions**
- **gst_incoherent : Non-coalesced (incoherent) global memory stores**
- **gst_coherent : Coalesced (coherent) global memory stores**
- **gst_32b : 32-byte global memory store transactions**
- **gst_64b : 64-byte global memory store transactions**
- **gst_128b : 128-byte global memory store transactions**
- **instructions : Instructions executed**
- **warp_serialize : Number of thread warps that serialize on address conflicts to either shared or constant memory**
- **cta_launched : Number of threads blocks executed**

CUDA Visual Profiler



cudaprof: Demo

- Profiling example code:
  `code2.cu`

- Profiling example code:
  `fdtd_cuda.cu`

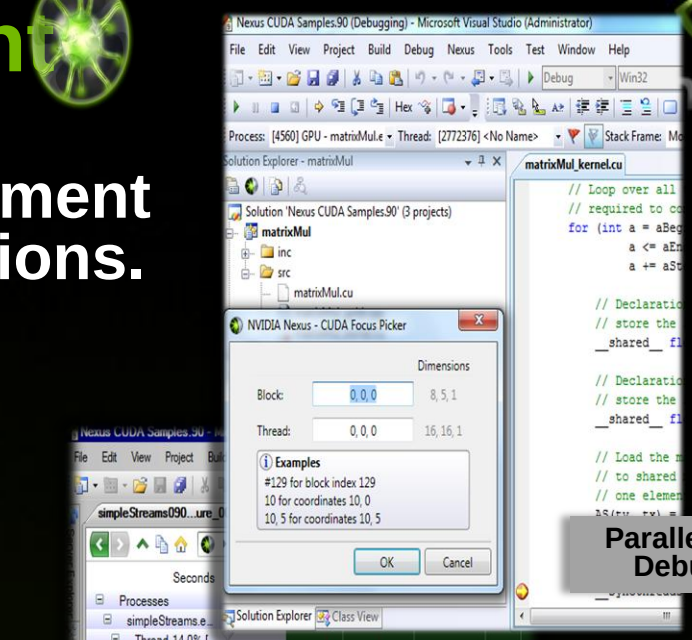
NVIDIA Parallel Nsight

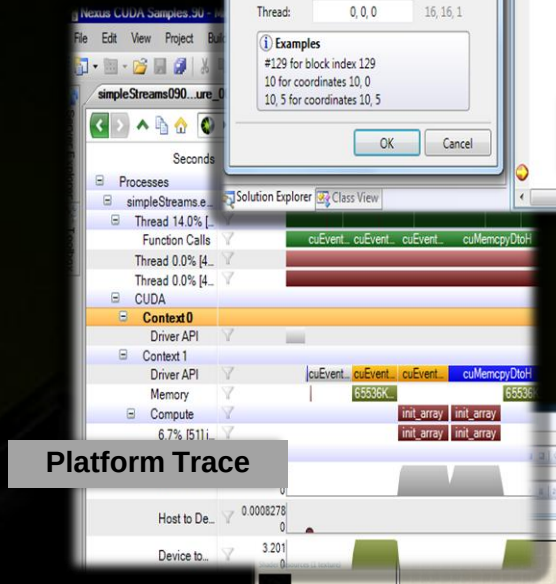
The first development environment for massively parallel applications.

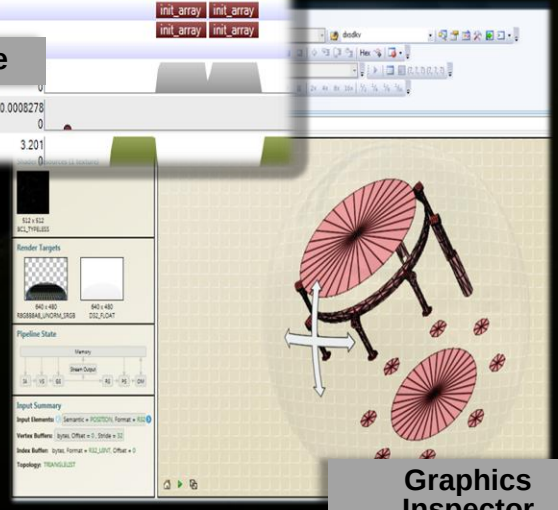
Hardware GPU Source Debugging

Platform-wide Analysis

Complete Visual Studio integration







http://developer.nvidia.com/object/nsight.html