

CUDA Efficient Programming

f.ficarelli@cineca.it



www.cineca.it



- 1. Overview and general concepts
- 2. Performance Metrics
- 3. Memory Optimizations
- 4. Execution Optimization
- 5. Tools Overview

Different worlds: host and device

	Host	Device
Threading resources	2 threads per core (SMT), 24/32 threads per node. The thread is the atomic execution unit.	e.g.: 1536 (thd x sm) * 14 (sm) = 21504. The Warp (32 thd) is the atomic execution unit.
Threads	«Heavy» entities, context switches and resources management.	Extremely lightweight, managed grouped into warps, fast context switch, no resources management (statically allocated once).
Memory	e.g.: 48 GB / 32 thd = 1.5 GB/thd, 300 cycles lat., 6.4 GB/s band (DDR3), 3 caching levels with lots of speculation logic.	e.g.: 6 GB / 21504 thd = 0.3 MB/thd, 600 cycles lat*, 144 GB/s band (GDDR5)*, fake caches. * coalesced



Maximum performance benefit

- Focus on achieving high occupancy.
- Focus on how to exploit the SIMT model at its best.
- Deeply analyze your algorithm in order to find the hotspots and embarassingly parallel-enabled portions.

i.e.: pay attention to the Amdahl's law, the porting could

be very tough.

$$S = \frac{1}{(1-P) + P/N}$$



Capability

The version tag that identifies:

- instructions and features supported by the board;
- coalescing rules;
- the board's resources constraints;
- througput of some instructions (hardware implementation).

Capability: atomics

SUNECA

ria Interunive

Feature Support/Unlisted features are	Compute Capability							
supported for all compute capabilities)	1.0	1.1	1.2	1.3	2.x, 3.0	3.5		
Atomic functions operating on 32-bit integer values in global memory (Atomic Functions)	No			Vec		-		
atomicExch() operating on 32-bit floating point values in global memory (atomicExch())								
Atomic functions operating on 32-bit integer values in shared memory (Atomic Functions)								
atomicExch() operating on 32-bit floating point values in shared memory (atomicExch())	 N	lo		Y	Yes			
Atomic functions operating on 64-bit integer values in global memory (Atomic Functions)								
Warp vote functions (Warp Vote Functions)								
Double-precision floating-point numbers	No				Yes			
Atomic functions operating on 64-bit integer values in shared memory (Atomic Functions)								
Atomic addition operating on 32-bit floating point values in global and shared memory (atomicAdd())	No			Yes				
ballot() (Warp Vote Functions)								
threadfence_system() (Memory Fence Functions)								
syncthreads_count(),								
syncthreads_and(),								
syncthreads_or() (Synchronization Functions)								
Surface functions (Surface Functions)								
3D grid of thread blocks								
Funnel shift (see reference manual)			No			Yes		

Conserver version of the run is a second sec

Technical Specifications	Compute Capability							
reclinical specifications	1.0	1.1	1.2	1.3	2.x	3.0	3.5	
Maximum dimensionality of grid of thread blocks			2			3		
Maximum x-dimension of a grid of thread blocks			65535			1		
Maximum y- or z-dimension of a grid of thread blocks				65535				
Maximum dimensionality of thread block				3				
Maximum x- or y-dimension of a block		5	12			1024		
Maximum z-dimension of a block				64				
Maximum number of threads per block		5	12			1024		
Warp size				32				
Maximum number of resident blocks per multiprocessor			8			_		
Maximum number of resident warps per multiprocessor	2	24	3	2	48 64			
Maximum number of resident threads per multiprocessor	70	68	10	24	1536	1536 2048		
Number of 32-bit registers per multiprocessor	8	К	16	δK	32 K	64	K	
Maximum number of 32-bit registers per thread		1	28		6	3	255	
Maximum amount of shared memory per multiprocessor		16	KB		48 KB		-	
Number of shared memory banks		1	6			32		
Amount of local memory per thread		16	KB		512 KB			
Constant memory size				64 KB				
Cache working set per multiprocessor for constant memory	8 KB							
Cache working set per multiprocessor for texture memory		Device d	ependent	, betweer	n 6 KB and	8 KB		
Maximum width for a 1D texture reference bound to a CUDA array		81	92			65536		



CUDA APIs

Runtime API	Handles kernel mechanics (loading, parameters setup, invocation, context management, fatbin management). Provides CUDA language extensions.
Driver API	Low level, pure C99 interface (nvcc not needed), explicit management of every resource (context, kernel params, etc)

The Driver API isn't forward compatible.





Performance metrics



Performance metrics

- Wall time
- Theroetical vs achieved bandwidth
- Achievable vs achieved occupancy
- Memory conflicts



Timing

- It's allowed to use std timing facilities (host side).
- Beware of asynchronous calls!

start = clock()
my_kernel<<< g, b, s >>>();
cudaThreadSynchronize();
end = clock();

- CUDA provides the *Events* facility.
- Needed to time single streams without loosing concurrency.



Bandwidth

- 1. Get board's theoretical bandwidth: $B = freq * rate = (1107 * 10^{6}) * \left(\frac{512 * 2}{8}\right) = 141.6 \ GB/s \quad \leftarrow \quad GeForce \ GTX \ 280$ $Clock \ freq. \ (MHz)$
- 2. Get kernel's effective bandwidth:

// __global__ device code, single precision data
if(threadIdx.x < 2048 && threadIdx.y < 2048) {
 mat_a[threadIdx.x] [threadIdx.y] = mat_b[threadIdx.x] [threadIdx.y];
}</pre>

$$B^* = \frac{D^r + D^w}{t} = \frac{2048^2 * 4 * 2}{t}$$

3. Measure kernel's achieved bandwidth: use profiling tools!

Beware of cudaprof: throughput result is extrapolated and considers wasted transaction data (uncoalesced) as good.



Memory Optimizations



Data Transfers

- Host and Device have their own address space
- GPU boards are connected to host via PCIex bus
- Low bandwidth, extremely low latency

Technology	Peak Bandwidth
PCIex GEN2 (16x, full duplex)	8 GB/s (peak)
PCIex GEN3 (16x, full duplex)	16 GB/s (peak)
DDR3 (full duplex)	26 GB/s (single channel)

Focus on how to minimize transfers and copybacks*.

* Try to find a good trade off!



Page-locked memory

•Pinned (or page-locked memory) is a main memory area that is not pageable by the operating system;

•ensures faster transfers (the DMA engine can work without raising interrupts);

•the only way to get closer to PCI peak bandwidth;

•allows CUDA asynchronous operations (including *Zero Copy*) to work correctly.

// allocate page-locked memory	// allocate regular memory
<pre>cudaMallocHost(&area, sizeof(double) * N);</pre>	area = (double*) malloc(sizeof(double) * N);
// free page-locked memory	// lock area pages (CUDA >= 4.0)
cudaFreeHost(area);	<pre>cudaHostRegister(area, sizeof(double) * N, cudaHostRegisterPortable);</pre>
	// unlock area pages (CUDA >= 4.0)
	<pre>cudaHostUnregister(area);</pre>
	// free regular memory
	cudaFreeHost(area):

Warning: locked pages are a limited resource (much smaller than regular pages, ulimit -1)



Zero Copy

 CUDA allows to map a page-locked host memory area to device's address space;

// allocate page-locked and mapped memory
cudaHostAlloc(&area, sizeof(double) * N, cudaHostAllocMapped);
// invoke retrieving device pointer for mapped area
cudaHostGetDevicePointer(&dev_area, area, 0);
my_kernel<<< g, b >>>(dev_area);
// free page-locked and mapped memory
cudaFreeHost(area);

- The only way to provide on-the-fly a kernel data larger than device's global memory.
- Very convenient for large data with sparse access pattern.

Unified Virtual Addressing

CUDA 4.0 introduced one (virtual) address space for all CPU and GPUs memory:

- automatically detects physical memory location from pointer value
- enables libraries to simplify their interfaces (e.g. cudaMemcpy)

Pre-UVA	UVA
Each source-destination permutation has its own option	Same interface
cudaMemcpyHostToHost cudaMemcpyHostToDevice cudaMemcpyDeviceToHost cudaMemcpyDeviceToDevice	cudaMemcpyDefault

Pointers returned by cudaHostAlloc() can be used directly from within kernels running on UVA enabled devices (i.e. there is no need to obtain a device pointer via cudaHostGetDevicePointer())



```
cudaDeviceCanAccessPeer(&can_access_peer_0_1, gpuid_0, gpuid_1);
cudaDeviceCanAccessPeer(&can_access_peer_1_0, gpuid_1, gpuid_0);
cudaSetDevice(gpuid_0);
cudaDeviceEnablePeerAccess(gpuid_1, 0);
cudaSetDevice(gpuid_1);
cudaDeviceEnablePeerAccess(gpuid_0, 0);
```

cudaMemcpy(gpu0_buf, gpu1_buf, buf_size, cudaMemcpyDefault);

•cudaMemcpy() knows that our buffers are on different devices (UVA), will do a P2P copy now

Note that this will *transparently* fall back to a normal copy through the host if P2P is not available



Multi-GPUs: direct access

global void SimpleKernel(float *src, float *dst)

```
const int idx = blockIdx.x * blockDim.x + threadIdx.x;
dst[idx] = src[idx];
```

```
cudaDeviceCanAccessPeer(&can_access_peer_0_1, gpuid_0, gpuid_1);
cudaDeviceCanAccessPeer(&can_access_peer_1_0, gpuid_1, gpuid_0);
cudaSetDevice(gpuid_0);
cudaSetDevice(gpuid_1);
cudaDeviceEnablePeerAccess(gpuid_0, 0);
cudaSetDevice(gpuid_0);
SimpleKernel<<<blocks, threads>>> (gpu0_buf, gpu1_buf);
SimpleKernel<<<blocks, threads>>> (gpu1_buf, gpu0_buf);
cudaSetDevice(gpuid_1);
SimpleKernel<<<blocks, threads>>> (gpu0_buf, gpu1_buf);
```

After P2P initialization, this kernel can now read and write data in the memory of multiple GPUs (just *deferencing pointers*!)
UVA ensures that the kernel knows whether its argument is from local memory, another GPU or zero-copy from the host



Asynchronous operations

- Kernel calls are asynchronous by default
- Memory transfers and copybacks are blocking
- The cudaMemcpy has an asynchronous version (cudaMemcpyAsync)
- Boards <= 1.3 can overlap copy-copy (opposite directions) and copykernel
- Boards >= 2.0 (Fermi and Kepler) can overlap kernel-kernel execution.





Streams

- A *stream* is a FIFO command queue;
- a stream is independent to every other active stream;
- CUDA streams are the main way to exploit concurrent execution and I/O operations.



cudaStream_t stream[3]; for (int i=0; i<3; ++i) cudaStreamCreate(&stream[i]);</pre>

float* hostPtr; cudaMallocHost((void**)&hostPtr, 3 * size);

for (int i=0; i<3; ++i) cudaMemcpyAsync(inputDevPtr+i*size, hostPtr + i * size, size, cudaMemcpyHostToDevice, stream[i]);</pre>

for (int i=0; i<3; ++i) myComputeKernel<<<100, 512, 0, stream[i]>>>(outputDevPtr + i * size, inputDevPtr + i * size, size);

for (int i=0; i<3; ++i) cudaMemcpyAsync(hostPtr + i * size, outputDevPtr+i*size, size, cudaMemcpyDeviceToHost, stream[i]);

cudaThreadSynchronize();

for (int i=0; i<3; ++i) cudaStreamDestroy(&stream[i]);</pre>





Streams: how to overlap kernels

Starting from capability 2.0 the board has the ability to overlap computations from multiple kernels where:

- submission of commands happens in a breadth-first fashion*;
- no synchronization happens between command stages;
- no operations occur on the default stream;
- the active streams are less than 16*;

*Kepler architecture introduced the *HyperQ* technology:

- No more need for breadth-first command submission
- Supports up to 32 concurrent streams



CUDA Memory Hierarchy

Grid								
Block (0, 0)		В	Block (1, 0)					
Shared	Memory		Sha	red	Memory			
Registers	Registers	F	Registers Registers					
Thread (0, 0) Thread (1, 0)			Thread (0, 0) Thread			d (1, 0)		
Local Memory	Local Memory	Î.	Local temory		Local Memory			
Global Memory								
Constant Memory								
Texture Memory								

Memory	Location on/off chip	Cached	Access	Scope	Lifetime
Register	On	n/a	R/W	1 thread	Thread
Local	Off	†	R/W	1 thread	Thread
Shared	On	n/a	R/W	All threads in block	Block
Global	Off	t	R/W	All threads + host	Host allocation
Constant	Off	Yes	R	All threads + host	Host allocation
Texture	Off	Yes	R	All threads + host	Host allocation



Global Memory

- Memory area with the same purpose as host's main memory;
- High(er) bandwidth, high(er) latency;
- In order to exploit its bandwidth at best, all accesses must be <u>coalesced</u>.



Optimizations : coalescing

- The global memory is accessed by 16 threads coalesced if the following three conditions are met:
- either 4-byte words, resulting in one 64-byte memory transaction
- Or 8-byte words, resulting in one 128-byte memory transaction
- Or 16-byte words, resulting in two 128-byte memory transactions
- All 16 words must lie in the same aligned segment
- Threads must access the words in a strictly increasing sequence: the nth thread in the half-warp must access the nth word.



Optimizations: coalescing, examples: OK





Optimizations: coalescing, examples: NON-OK



Permuted Access by Threads

Non sequential memory access, resulting in 16 memory accesses



Misaligned Starting Address (not a multiple of 64)

The starting address is misaligned, the result is 16 memory accesses



Optimizations: coalescing (5), examples: NON-OK

Non-contiguos access will result in 16 memory accesses





Coalescing for Capability 1.2

The memory controller of 1.2 cards is much improved, access as that in figure would occur in a single transaction





Coalescing for Capability 1.2

Random access within a segment: single 64B transaction (if they fit)





Capability 1.2

Hits misaligned, anyway, occur in a single transaction





Capability 1.2

Misaligned accesses, ending in 2 different 64B segments occur in two transactions





Coalescing for latest architectures

The memory controller has been vastly improved:

On devices with capability >= 2.0, memory accesses by the threads of a warp** are coalesced into the minimum number of L1 lines* that satisfies all threads.

*L1: 128B-aligned, 128B wide lines. ***half-warp* based for previous architectures.

Coalescing: examples

			Ali	igned and se	quent	tial		
Addresses:	96	12	3	160	192	224	256	288
Threads:	_	† 0	111				1111 31	
Compute	capabi	lity:	1.	0 and 1.1	1.	2 and 1.3	2.x and	3.0
Memory tr	ansacti	ons:		Unca	ched		Cach	ed
			1 x 1 x	64B at 128 64B at 192	1 x 1 x	64B at 128 64B at 192	1 x 128B	at 128

NECA

		Alig	ned and non-	sequer	ntial		
Addresses:	96	128	160	192	224	256	288
Threads:		ţ††	X11111111		111111111111	1111 31	
Compute	capabil	ity: 1	.0 and 1.1	1.2	and 1.3	2.x and	3.0
Memory tr	ansactio	ons:	Unca	ched		Cache	ed
		8 x 8 x 8 x 8 x	32B at 128 32B at 160 32B at 192 32B at 224	1 x 1 x	64B at 128 64B at 192	1 x 128B	at 128

		Misa	aligned and	l sequent	ial		
Addresses:	96	128	160	192	224	256	288
Threads:		1111			1111111	///// 31	
Compute	capabil	lity: 1.	0 and 1.1	1.2	and 1.3	2.x and	3.0
Memory tr	ansactio	ons:	Un	cached		Cache	ed
		7 x 8 x 8 x 8 x 1 x	32B at 12 32B at 16 32B at 19 32B at 22 32B at 25	8 1 x 12 0 1 x 6 2 1 x 3 4 6	8B at 128 4B at 192 2B at 256	1 x 128B 1 x 128B	at 128 at 256



Shared memory

- A sort of *explicit* cache;
- resides on the chip so it is *much* faster than the onboard memory;
- size is 16KB (48KB on Fermi by default*)
- 16 (32 for Fermi) banks can be accessed simultaneously by the same warp;
- Banks are organized such that:
 - successive 32-bit words are assigned to successive banks;

each bank has 32-bit per cycle bandwidth.

*Kepler architecture introduced some improvements:

 ability to switch from 4B to 8B banks (2x bandwidth for double precision codes)



Shared Memory: bank conflicts

If at least two threads belonging to the same half-warp (whole warp for capability 1.0) access the same shared memory bank, then the accesses are serialized (groups transactions in conflict-free accesses);

 If all the threads access the same address, a *broadcast* is performed;

If part of the half-warp accesses the same address, a *multicast* is performed (capability >= 2.0);



Texture memory

- Load requests are cached;
- it is read only, must be set by the host;
- could bring benefits if the threads within the same block access memory using regular 2D patterns, but you need appropriate binding;
- specifically, texture memories and caches are designed for graphics applications where memory access patterns exhibit a great deal of spatial locality;
- dedicated hardware for on-the-fly interpolation.

For typical linear patterns, global memory (if coalesced) is faster.



Texture Memory

```
// allocate array and copy image data
cudaChannelFormatDesc channelDesc =
                    cudaCreateChannelDesc(32, 0, 0, 0, cudaChannelFormatKindFloat);
cudaArray* cu array;
cudaMallocArray( &cu array, &channelDesc, width, height );
cudaMemcpyToArray( cu array, 0, 0, h data, size, cudaMemcpyHostToDevice);
// set texture parameters
tex.addressMode[0] = cudaAddressModeWrap;
tex.addressMode[1] = // declare texture reference for 2D float texture
tex.filterMode = cu( texture<float, 2, cudaReadModeElementType> tex;
tex.normalized = tru
// Bind the array to global void
cudaBindTextureToAr: transformKernel( float* g odata, int width, int height, float theta)
                      // calculate normalized texture coordinates
                      unsigned int x = blockIdx.x*blockDim.x + threadIdx.x;
                      unsigned int y = blockIdx.y*blockDim.y + threadIdx.y;
                      float u = x / (float) width;
                      float v = y / (float) height;
                      // transform coordinates
                      u = 0.5f;
                      v = 0.5f;
                      float tu = u^{*}cosf(theta) - v^{*}sinf(theta) + 0.5f;
                      float tv = v*cosf(theta) + u*sinf(theta) + 0.5f;
                      // read from texture and write to global memory
                      q odata[y*width + x] = tex2D(tex, tu, tv);
```

Kepler: global loads through texture

The compiler (LLVM) can detect texture-compliant loads and map them to the new *«global load through texture»* PTX instruction:

- global loads are going to pass through texture pipeline;
- dedicated cache (no L1 pressure) and memory pipe, relaxed coalescing;
- automatically generated by compiler (no texture map needed) for accesses through compliant pointers (constant and restricted);
- useful for bandwidth-limited kernels (bandwidths sum).



Constant Memory

- Extremely fast on-board memory area
- Read only, must be set by the host
- 64 KB, cached reads in a dedicated L1 (register space)
- Coalesced access if all threads of a warp read the same address
- Useful to off-load long argument lists from shared memory

```
__device____constant__ parameters_t args;
__host__ void copy_params(const parameters_t* const host_args)
{
    cudaMemcpyToSymbol("args", host_args, sizeof(parameters_t));
}
```



Registers

- Just like CPU registers, access has no latency;
- used for scalar data local to a thread;
- taken by the compiler from the SM pool (32K for Fermi, 64K for Kepler) and statically allocated to each thread;

 register pressure one of the most dangerous occupancy limiting factors.



Registers

Some tips:

- try to fold "stack" variables (it would be less useful on LLVM)
- try to offload data to shared memory;
- use launch bounds to force the number of resident blocks;

```
#define MAX_THREADS_PER_BLOCK 256
#define MIN_BLOCKS_PER_MP 2
___global___void
__launch_bounds__(MAX_THREADS_PER_BLOCK,
MIN_BLOCKS_PER_MP)
```

- my_kernel(int* inArr, int* outArr) { ... }
- Imit register usage via compiler option.

```
# nvcc -Xptas -v mykernel.cu
ptxas info : Compiling entry function '_Z12my_kernelP9domain_t_' for 'sm_20'
ptxas info : Used 13 registers, 8+16 bytes smem
```

```
# nvcc --maxrregcount 10 -Xptas -v mykernel.cu
ptxas info : Compiling entry function '_Z12my_kernelP9domain_t_' for 'sm_20'
ptxas info : Used 10 registers, 12+0 bytes lmem, 8+16 bytes smem
```



Local memory and caches





Local memory

- "Local" because it's private on a per-thread basis;
- it's actually a global area used to spill out data when SM runs out of resources;
- addressing is resolved by the compiler;
- cached (store only).



Caches

 FERMI architecture introduces caching mechanisms for global memory accesses (constant and texture are cached since 1.0)

 L1: private to thread, virtual cache implemented into shared memory

// L1 = 48 KB
// SH = 16 KB
cudaFuncSetCacheConfig(kernel, cudaFuncCachePreferL1);
// L1 = 16 KB

// SH = 48 KB

cudaFuncSetCacheConfig(kernel, cudaFuncCachePreferShared);

// Try to decrease spilled registers eviction from L1,// disable L1 caching for global memory loads\$ nvcc -Xptas -dlcm=cg

*Kepler architecture introduced some improvements:
New 32 KB + 32 KB partition option

L2: 768KB, grid-coherent, 25% better latency than DRAM



Execution Optimization



Occupancy

The board's occupancy is the ratio of active warps to the maximum number of warps supported on a multiprocessor.

Keeping the hardware busy helps the warp scheduler to hide latencies.



Occupancy: constraints

Every board's resource can become an occupancy limiting factor:

- shared memory;
- grid and block sizes;
 - (max threads per SM/max blocks per SM)
- used (and spilled) registers

Given an actual kernel configuration, is possible to predict the maximum *theoretical occupancy* allowed.



Occupancy: block sizing tips

Some experimentation is required.

However there are some heuristic rules:

- threads per block should be a multiple of warp size;
- a minimum of **64 threads per block** should be used;
- 128-256 threads per block is universally known to be a good starting point for further experimentation;
- prefer to split very large blocks into smaller blocks.

Kepler: dynamic parallelism

• One of the biggest CUDA limitations is the need to fit a single grid configuration for the whole kernel.

If you need to reshape the grid, you have to resync back to host and split your code.

- Kepler (in addition to CUDA 5.x) introduced *Dynamic Parallelism*
- It enables a global kernel to be called from within another kernel
- The child grid can be *dynamically sized* and *optionally synchronized*





Instructions

Arithmetic ops:

- prefer integer shift operators instead of division and modulo (would be less useful with LLVM);
- beware of (implicit) casts (very expensive);
- use intrinsics for trascendental functions where possible;
- try the fast math implementation.

Capability: instruction throughput

	Compute Capability					
	1.0 1.1 1.2	1.3	2.0	2.1	3.0	3.5
32-bit floating-point add, multiply, multiply-add	8	8	32	48	192	192
64-bit floating-point add, multiply, multiply-add	1	1	16(*)	4	8	64
32-bit integer add	10	10	32	48	160	160
32-bit integer compare	10	10	32	48	160	160
32-bit integer shift	8	8	16	16	32	64
Logical operations	8	8	32	48	160	160
32-bit integer multiply, multiply-add, sum of absolute difference	Multiple instructions	Multiple instructions	16	16	32	32
24-bit integer multiply ([u]mul24)	8	8	Multiple instructions	Multiple instructions	Multiple instructions	Multiple instructions
32-bit floating-point reciprocal, reciprocal square root, base-2 logarithm (log2f), base 2 exponential (exp2f), sine (sinf), cosine (cosf)	2	2	4	8	32	32
Type conversions from 8- bit and 16-bit integer to 32-bit types	8	8	16	16	128	128
Type conversions from and to 64-bit types	Multiple instructions	1	16(*)	4	8	32
All other type conversions	8	8	16	16	32	32
(*) Throughput is lower for GeForce GPUs.						

ADEK



Control Flow

Different execution paths inside the same warp are managed by the predication mechanism and lead to thread divergence.

if (threadIdx.x == 0) $\{...\}$

if (threadIdx.x == 0) {...} else {...}

if (threadIdx.x == 0) {...} else if (threadIdx.x == 1) {...}

if (vec[threadIdx.x] > 1.0f) $\{...\}$

- Minimize the number of execution branches inside the same warp;
- make the compiler's life easier by <u>unrolling</u> loops (hand-coded, pragma or option);
- use signed counters for loops (would be less useful with LLVM);

Exploiting Multi-GPUs

CUDA >= 4.0 introduced the N-to-N bound feature:

- 1. Every thread can be bound to any board
- 2. Every board can be bound to an arbitrary number of threads

Multi-GPU can be exploited through your favourite multithreading paradigm (OpenMP, pthreads, etc...)

```
#pragma omp parallel
#pragma omp sections
{
    #pragma omp section
    {
        cutilSafeCall(cudaSetDevice(0));
        cudaMemcpy(device_data_1, host_data_1, size, cudaMemcpyHostToDevice);
        my_kernel<<< grid, block >>>(device_data_1);
        // ...
    }
    #pragma omp section
    {
        cutilSafeCall(cudaSetDevice(1));
        cudaMemcpy(device_data_2, host_data_2, size, cudaMemcpyHostToDevice);
        my_kernel<<< grid, block >>>(device_data_2);
        // ...
    }
}
```



Tools Overview



- Common
 - Memory Checker
 - Built-in profiler
 - Visual Profiler
- Linux
 - CUDA GDB
 - Parallel Nsight for Eclipse
- Windows
 - Parallel Nsight for VisualStudio

Profiling tools: built-in

The CUDA runtime provides a useful profiling facility without the need of external tools.

export CUDA_PROFILE=1 export CUDA PROFILE CONFIG=\$HOME/.config

// Contents of config
gld_coherent
gld_incoherent
gst_coherent
gst_incoherent

E. TAY

gld_incoherent: Number of non-coalesced global memory loads gld_coherent: Number of coalesced global memory loads gst_incoherent: Number of non-coalesced global memory stores gst_coherent: Number of coalesced global memory stores local_load: Number of local memory loads local_store: Number of local memory stores branch: Number of branch events taken by threads divergent_branch: Number of divergent branches within a warp instructions: instruction count warp_serialize: Number of threads in a warp that serialize based on address conflicts to shared or constant memory cta launched: executed thread blocks

```
method,gputime,cputime,occupancy,gld_incoherent,gld_coherent,gst_incoherent,gst_coherent
method=[ memcopy ] gputime=[ 438.432 ]
method=[ Z17reverseArrayBlockPiS_ ] gputime=[ 267.520 ] cputime=[ 297.000 ] occupancy=[ 1.000 ]
gld_incoherent=[ 0 ] gld_coherent=[ 1952 ] gst_incoherent=[ 62464 ] gst_coherent=[ 0 ]
method=[ memcopy ] gputime=[ 349.344 ]
```

Profiling: Visual Profiler

- Traces execution at host, driver and kernel levels (unified timeline)
- Supports automated analysis (hardware counters)

File View Ran Help												
		1										
C *dct8x8.vp II						Propert	ties 32 📷	Detail Graph	s			
	161.7 ms	161.8 ms	File View Run Help									
Process: 11119				Q Q A J	£.							
Runtime API Driver API			🗣 *diverge.vp 🗵							·· 🗆 🖬	Properties 🖬 Detail Graphs 32	· 🗆
□ [0] GaForce GTX 480 □ Contast 1 (CUDA) □ MemCpy (HoD) □ MemCpy (DtoD) □ MemCpy (DtoD) □ Compute □ 0 7% [101] CUDAk □ 0 7% [101] CUDAk □ 0 7% [10] CUDAk □ 0 7% [1] CUDAke	CUDAkeme	IOua CUDAken CUDAken	Partime AP Driver AP Eliol Geforce GTK 430 ■ Context 1 (GUDA) ▼ MemCpy (HtoD) ▼ MemCpy (HtoD) ■ Compute ▼ 42.5% (4) Vec1of320 ▼ 7.5% (4) Vec1of320 ▼ 7.4% (4) Vec1of320 ▼ 7.4% (4) Vec1of320 ▼ 7.4% (4) Vec1of320 ▼ 7.4% (4) Vec1of320 ▼ 5.4% (4) Vec1of320 ▼ 5.5% (4) Vec1of320 ▼ 5.5% (4) Vec1of320 ▼ 5.5% (4) Vec1of320 ■ Stream 1		0.1 s		VecTher	vecSolin VecSolin VecSolin	15 s 10 In., Vec 30/32/01 Vec 30/32/01 10 In., Vec 30/32/01		Max: 95.261 ms 17.85 ms	10 H 30
	a		🖬 Analysis 📰 Details 🕄 🕮 (Console 🔳 Se	ttings						2 G 5	
Timeline Multiprocessor Kernel Memory Kernel Instruction	9yzz All O O O	Analysis Results High Branch Div Divergent branche High Instruction A combination of c High Global Mer Non-coalesced glo	VecThenting*, ing*, ing*, ing VecS0(ing*, ing*, ing) VecS0(ing*, ing*, ing) VecS0(ing*, ing*, ing) VecS0(ing*, ing*, ing*, ing) VecS0(ing*, ing*, ing*, ing) VecS0(ing*, ing*, ing*, ing)	Properties 32 Detail Ge View Ran Help Properties 33 Properties 33 Detail Ge View Ran Help Properties 41 Properties 43 Propetis 44 Propetis 44 <th< th=""><th>16 16 16 16 16 2 16 16 16</th><th>41.05 MB/s 40.51 MB/s 40.51 MB/s 40.51 MB/s 40.51 MB/s 40.51 MB/s 0.8/s 81 MB/s 81.124 MB/s 81.16 MB/s</th><th>369.46 KB 367.79 KB 40.87 KB 40.87 KB 40.87 KB 40.87 KB 0 B 0 B 0 B 0 B 0 B</th><th>12.5% 12.5% 12.5% 12.5% 12.5% 12.5% 12.5% 12.5% 12.5% 12.5%</th><th></th></th<>	16 16 16 16 16 2 16 16 16	41.05 MB/s 40.51 MB/s 40.51 MB/s 40.51 MB/s 40.51 MB/s 40.51 MB/s 0.8/s 81 MB/s 81.124 MB/s 81.16 MB/s	369.46 KB 367.79 KB 40.87 KB 40.87 KB 40.87 KB 40.87 KB 0 B 0 B 0 B 0 B 0 B	12.5% 12.5% 12.5% 12.5% 12.5% 12.5% 12.5% 12.5% 12.5% 12.5%				
	0.94		Vec1af32x(int*, int*, int*, int)	104.287 ms	7.717 ms	[2,1,1]	[1,1,1]	16	43.76 MB/s	0 B/	12.5%	



- Well-known tool enhanced with CUDA extensions
- Works well on single-gpu systems (OS graphics disabled)
- Can be run under GDB-targeted tools and GUIs (multigpu systems)

```
(cuda-gdb) info cuda threads
BlockIdx ThreadIdx To BlockIdx ThreadIdx Count Virtual PC Filename Line
Kernel 0* (0,0,0) (0,0,0) (0,0,0) (255,0,0) 256 0x00000000866400 bitreverse.cu 9
(cuda-gdb) thread
[Current thread is 1 (process 16738)]
(cuda-gdb) thread 1
[Switching to thread 1 (process 16738)]
#0 0x000019d5 in main () at bitreverse.cu:34
34 bitreverse<<1, N, N*sizeof(int)>>>(d);
(cuda-gdb) backtrace
#0 0x000019d5 in main () at bitreverse.cu:34
(cuda-gdb) info cuda kernels
Kernel Dev Grid SMs Mask GridDim BlockDim Name Args
0 0 1 0x0000001 (1,1,1) (256,1,1) bitreverse data=0x110000
```

Debugging: CUDA-MEMCHECK

- It's able to detect buffer overflows, misaligned global memory accesses and leaks
- Device-side allocations are supported
- Standalone or fully integrated in CUDA-GDB

```
$ cuda-memcheck --continue ./memcheck demo
====== CUDA-MEMCHECK
Mallocing memory
Running unaligned kernel
Ran unaligned kernel: no error
Sync: no error
Running out of bounds kernel
Ran out of bounds kernel: no error
Sync: no error
======= Invalid global write of size 4
====== at 0x00000038 in memcheck demo.cu:5:unaligned kernel
======= by thread (0,0,0) in block (0,0,0)
====== Address 0x200200001 is misaligned
_____
======= Invalid global write of size 4
======= at 0x00000030 in memcheck demo.cu:10:out of bounds kernel
======= by thread (0, 0, 0) in block (0, 0, 0)
======= Address 0x87654320 is out of bounds
_____
_____
====== ERROR SUMMARY: 2 errors
```

ADE.

Parallel NSight

- Plug-in for major IDEs (Eclipse and VisualStudio)
- Aggregates all external functionalities:
 - Debugger (fully integrated)
 - Visual Profiler
 - Memory correctness checker
- As a plug-in, it extends all the convenience of IDEs to CUDA

On Windows systems:

- Now works on a single GPU
- Supports remote debugging and profiling
- Latest version (2.2) introduced live PTX assembly view, warp inspector and expression lamination

E.S.

Parallel NSight

vanelpi	pe_demo	orita (D	ebugging) - Mic	nosoft Visual S	Rudio (Administ	nator)										perl-t	-
Ein Enbl	Date	Injact	Build Debug	Team Ma	ight Dgta Is	sola Tegt And	yza Winds	w Halp	11 Mail Sub Draws	1000		La Labora de		e mina	2.000 (22.020)		
				-			1.14	a service states	Storight.		THE OWNER AND ADDRESS			COUR	and any or	- F.	
Process	[1340] w	averpipe	e_demo.exe.	- Diread	128/49121 <14	o Name>	1.1	 Stack Prame: 	COmposite (COORER) -]	l trace - Line 148							
Con	nections	localh	ost		4												
CUENA Info	01											+ + ×	CUD	A. WierpW	ratch1		* + >
00	Warps		· • • •	V fibe								Viewing 128/128	100	There	ested.		
	(model and		and in the second				-						2	Name	thy_max	🖗 nay jawa	
Currenti	blackder		Warp Index	PC	Active Ments	Shahun	Exception	Fils Norror	Source Lin; Lenes			110	4 3	Туре	_local_float	_local_float	_10
	(0,	ð,	e) (0 0x003elad	Querterreso	Breakpolst	Norre	rt_render.ou	363	1988年2月1日日日の1988年1	医结核菌毒结核菌 建苯	5百姓 日本		a	-1.4444908	·1.7955524	-21
	(B,	ø,	e): :	1 Buddleladi	Battititag	Breakpoint	Note	rt_render.cu	163		822232323333		1.1	1	-1,44425	-1.7907783	-21
	(B,	ø,	a) 1	0.0003eladi	Gerttttt:20	 Breakpolnt 	None	rt_render.cu	165	1986 p 6 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2		2 12 10 10 12 12		<u> 3</u>	-1.4440092	-1.7980076	-23
	ί ΰ,	ο,	4) 1	0x003wladi	Qufffffson	None	Note	rt_render.cs	265					3	-1.4437686	·1.7992405	-2,1
	(1,	٥,	d) (0 Robbiel290	0x23e00000	Breakpoint	Norm	rt_render.cu	148		민준아 바라 - 생활원 전 1			4	-1.4435281	-1.800477	-21
	(I,	Φ,	a)	0x0023w1200	9x87c99888	Breakpoint	None	rt_render.cu	148			문화 학학왕		2	-1.4432676	-18017174	-2,1
	[1,	ø,	a) :	1 BobbSecke70	Qufffffffff	Hone	None	ri_include.h	423				127		-1.4430474	-1,8029615	-21
						~			and the second		necessaries and the second		1 - 7	7	-1.4428074	-1.8042094	
		1					_							8	-1.4425675	-1.8054006	-2.1
CODAIN	101	Module	£											9	-1.4423276	-1.8007161	-2.1
rt, render.	cu 100								 Disastembly 			* * *	a - 1	10	-1.4420678	-1,8079749	-2.0
Hinder and	- Frank	-			100				Address			-11		11	-1.4438485	-18092378	-2.1
TURNOVOW	in scoper	-			-	1.			-	NACOT .			1-1	12	-1.4410069	-18105048	-2.1
140		100	node index	- node.get	Tupex(): ()	lineb to turn	0.		T - Viewing S	popas	N N N S AN AN AN	NY 0510 NY	4	13	-1.4413697	-18117749	-2.3
145		1							143:	const uint	t32 leaf_index = n	ode.get_index(14	-1.4411306	-1.8130492	-21
3.40		1							0x007412	a 1000400010210584	MIN HE, E[BORN]]	ase 11		15	-1.4408917	-1,8143274	-2.1
147		- 52	11-leaf int	tersection					0:003412	a independence index	1474 R7, R11			16	-1.4406527	-1.8150093	-2,1
2 148			penst uint:	52 leaf ind	ex = node.ge	t index();			9x091x12	@ 250000015210de4	MON RG. RG:			17	-1.4404343	·1.8168953	-2.1
140			const Byh_	leaf leaf	- geometr	y.m boh leave	s[leaf_in	idex];	0x003e12	4801000018411:03	LADO RA.CC. RW.	80;		15	-1/4401754	-1,818185	-21
138	9		const uint:	52 leaf_end	- leef.ge	t_index() + 1	eaf.get_s	120();	0x003e12	480000001:515:43	IADD.X RS, RS,	871		19	-1.439937	-1.8194788	-2.1
191			const wint!	52 leaf_beg	in = leaf.ge	t_index();			9x093x12	3 3000000010211de4	107V R.F. R.41			20	-1,4390986	-3.820776	-2.1
152							1411423	2011/2011/2012	Dx093e12	10 3.000000014015sis4	MOV R5, R51			21	-1.4394605	-1.8220775	-2.1
133	100		tur (uinti)	2 tri index	- Test pest	n; tri index	C Lewit and	1; ++tr1_1nbe	Deficite12	11 2500000014015de4	MOV ES, 451;	19	4.27	22	-1,4392225	-1.8233831	-2.1
100 75	1.1								1	2		1.4	1 27	23	-1.4389844	-1.8240926	-21
Locati								Die Call Strick				+ # X		24	-1.4387469	-1.8200059	-7.1
Name			-	Value			Type	+ Name	Same and the second second	100000		Language 4		25	-1.4385092	-1.8273233	-2.1
H Q L	af			Im size = 67	106175 m under	= 01	local	Clime	dule 05508fe0 - 121 trace	- Line 148		CUDA		26	-1.4382718	-1.828645	-2.1
94	set index			Test index h	as no value at t	he target location	-	Clima	dule 055981e0 - [1] rend	r givel - Line 409	8	CUDA		27	-1.4380344	-18299706	-2.1
	of and			leaf and he	s no value at the	eterget location.		Climo	dule 05508fe0 - (0) rt. tra	ce_primary_kernel - Line 423	1	CUDA		28	-1,4377974	-1.8313001	-21
9.1	saf, begin			Test begin'l	as no value at t	he target location	S ()	1				- 1122025		29	-1.4375603	·1.832634	-21
H) @ n	cde			[m_packed_o	data = 21.474848	77, m_skip_node :	24 _local	1.00						30	-1.4373236	-1.8339716	-21
H. 0	T21669			1= -1.43946	05, y = -1.82207	75, I = -2.1.50774)	_local	12						31	-1.4370668	-1.8353136	-2.1
H 9 n	njinv			(a = -1.43046	05, y = -1.82207	75; z = -2.150774)	_local						1			U	
9 0	ade_inde	č		'node_inder'	has no value at	the target location	n	14				10	CIE	0.4 Warm	Watch 1	utaut.	
and the second				11 3 51933		110 0.301 1.111	222 mileste						all some	to a to a to	and the second se	262	
leady-																	

ADEK

orala Interuniw

Parallel NSight

	Debug	- Win32 - 🙆 RundmelpiTracet - 🖓 🕆 🖬 🖄 🖬 🐮 👘 👘 🛓	
genonicSied1111pture_000.nvreport × Activity1.	niact**		
🕘 🗶 🌒 🔸 Temekne			
e Fiters 🕼			
	Time	0 41 42 43 44 45 44 47 48 44 41 48 44 45 44 15 14	35 4
0.0 % [71] fluid_advect/velocity_k	1.1		
0.0 % [70] fluid_diffuseProject_k	12		
0.0 % [70] fluid_update//elocity_k	¥		
0.0 % [70] fluid_updateParticles_k	V.		
0.0 % [71] Ruid_offsetVelocities_k			
0.0 % [284] fluid_setVelocity_k	1.4		
n Streams			
= DV			
Frames	0.	773 * * * * * * * * * * * * * * * * * *	
B Desice Context 8x3C28000	6.303600		
Device 0x200	V		STATISTICS.
Draw Calls	×.		
System			
CPU Usage		And a state of the second	and the second
Care 0		ALLA INCOME. AND ALLAND AND AND AND AND AND AND AND AND AND	
Contl		Addition the state A	-
Scie 2		LARGER ARE IN TO ANTIONNA ARE A AREA . A REALISTIC AND A . A ARE . ARE . ARE .	1.0.00
Row Information	Time Report		
Draw Calls ID restX Draw Calls Road	Start	Before Cantorn	
	End	After Capture	
CALIFY AND THE REAL	Duration		
# 161491 (Direct® Draw Call Workbad)	System		
5850 [DirectR Command Buffer Workload]	Process	Sustersonicsted eve [5812]	
# Drawbidexect (Direct% Draw Auriction Cell)	# Threads	32	
 0x3C25CCD (Divertit Device Context) 	Conmand Une	"citempisdomine-H4/c/program files (#86/unida corporation/unida demos/supersonic sled/bin/supersonicsled.exe"	

www.cineca.it

ADER

Consorale Interuniv