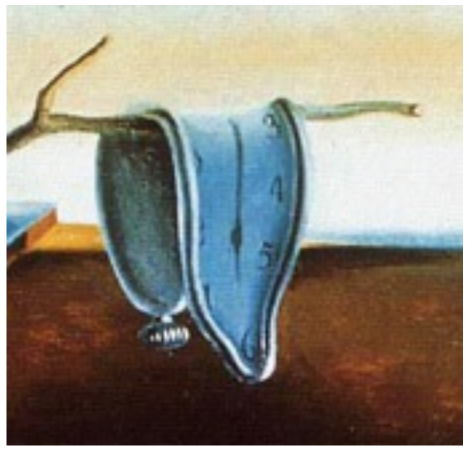
The Performance of Memory:

Avoiding time-wasting pitfalls



Fall 2012

CITA|ICAT

HW2

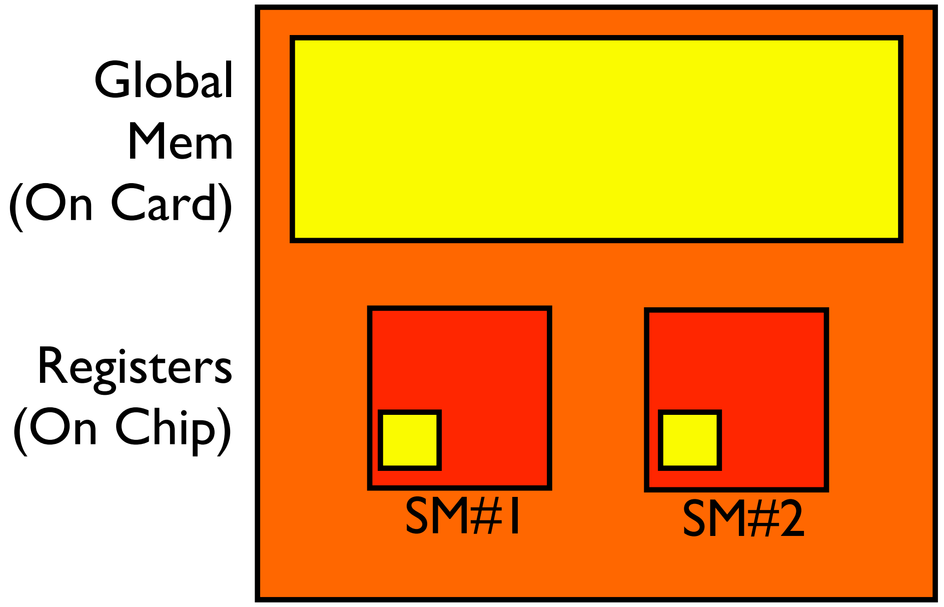
```
uint8_t charmean(uint8_t *data, const int r, const int c, const int

    const double sig = 2./9.*(nhalo*nhalo);
    double pi = 4.*atan(1.);
    double mean=0.;
    for (int i=r-nhalo; i<=r+nhalo; i++) {
        for (int j=c-nhalo; j<=c+nhalo; j++) {
            double d  = 1.0*data[i*rowsize+j];
            double e = expf(-(double)((i-r)*(i-r)+(j-c)*(j-c))/sig)
            mean += d*e;
        }
    }

    mean /= sig*pi;

    return (uint8_t)roundf(mean);
}
```

HW2

```
__global__ void cuda_smoothimage(uint8_t *data,
                                 uint8_t *smootheddata,
                                 const int nhalos, const int rows, const int cols) {

    const double sig = 2./9.*(nhalos*nhalos);
    const double pi = 4.*atan(1.);
    const int rowsize=cols+2*nhalos;

    int r = threadIdx.y + blockIdx.y*blockDim.y;
    int c = threadIdx.x + blockIdx.x*blockDim.x;

    if (r<rows && c<cols) {
        r += nhalos;
        c += nhalos;
        double mean = 0.;
        for (int i=r-nhalos; i<=r+nhalos; i++) {
            for (int j=c-nhalos; j<=c+nhalos; j++) {
                double d  = 1.0*data[i*rowsize+j];
                double e = exp(-(double)((i-r)*(i-r)+(j-c)*(j-c))/sig);
                mean += d*e;
            }
        }
        smootheddata[r*rowsize + c] = (uint8_t)round(mean/(sig*pi));
    }
    return;
}
```

CUDA Memories

Memory	On Chip?	Cached?	R/W	Scope
Register	On	No	R/W	**Thread**
Shared	On	No	R/W	**Block**
Global	Off	No	R/W	Kernel, Host
Constant	Off	Yes	R	Kernel, Host
Texture	Off	Yes	R(W?)	Kernel, Host
'Local'*	Off	No	R/W	Thread



* if you run out of registers, will put 'local' mem in global.

CITA|ICAT

SciNet

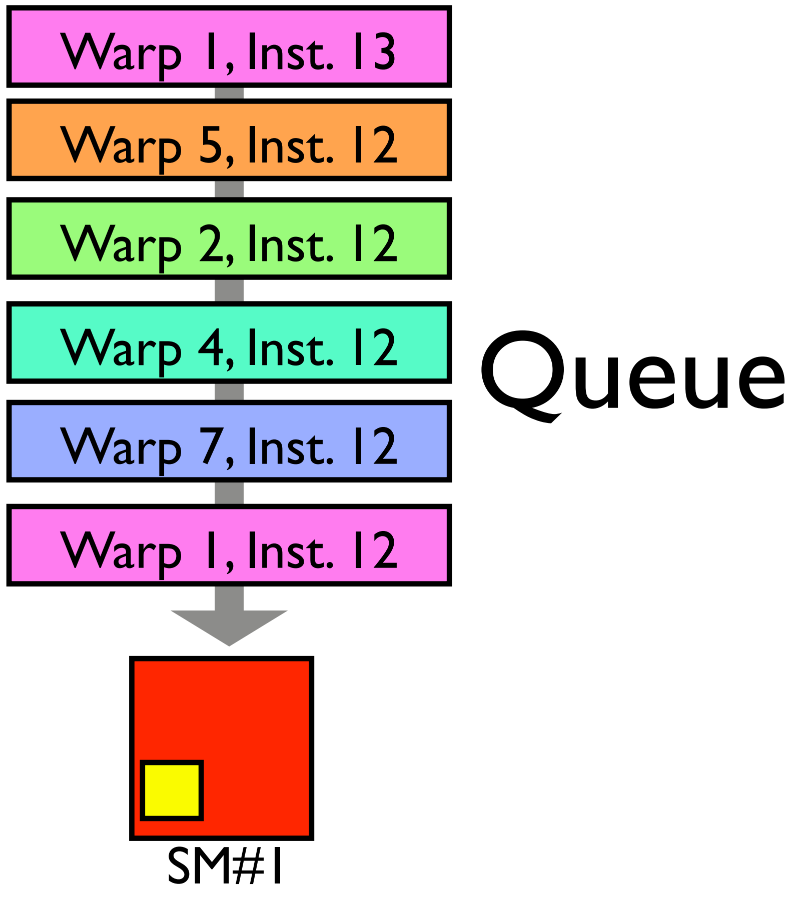
Making effective use of CUDA memories

- Preload data wherever possible
- Global memory -
  - Coalesced access
  - Make use of 128B (or, maybe, 32B) at a time
- Profiler to see what's happening
- Shared memory
  - Bank conflicts

Memory	On Chip?	Cached?	R/W	Scope
Register	On	No	R/W	**Thread**
Shared	On	No	R/W	**Block**
Global	Off	No	R/W	Kernel, Host
Constant	Off	Yes	R	Kernel, Host
Texture	Off	Yes	R(W?)	Kernel, Host
'Local'*	Off	No	R/W	Thread

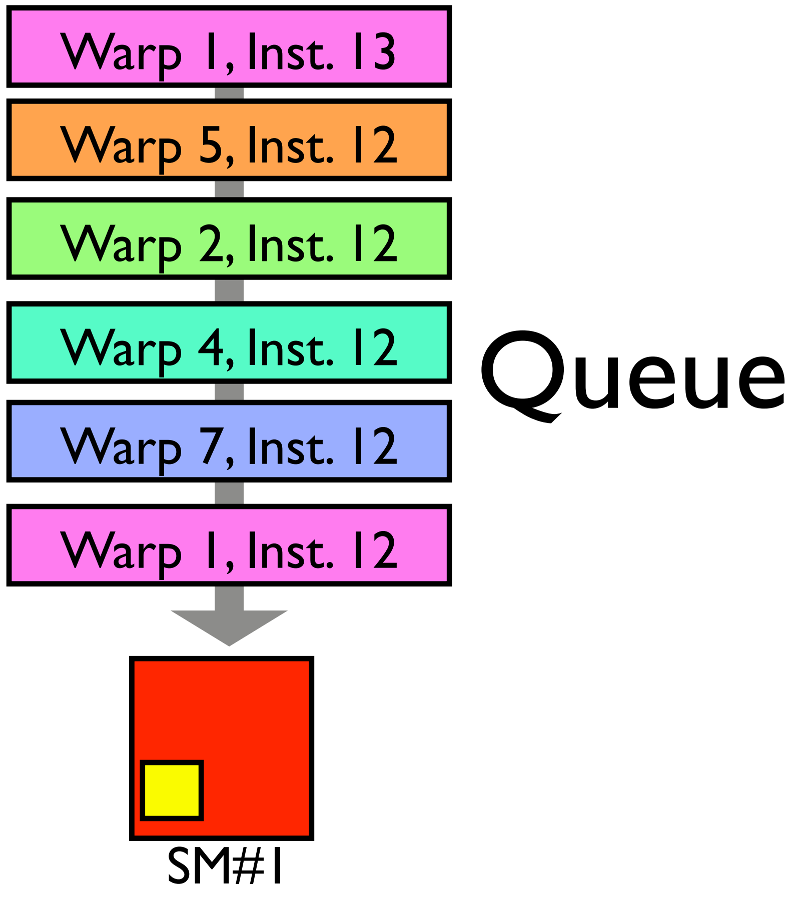
Stalling on Memory Access

- Graphics card schedules by the warp on an SM
- All warps that are ready to execute get scheduled
- Not ready to execute - stalled on memory access
- Nothing ready - SM sits idle.



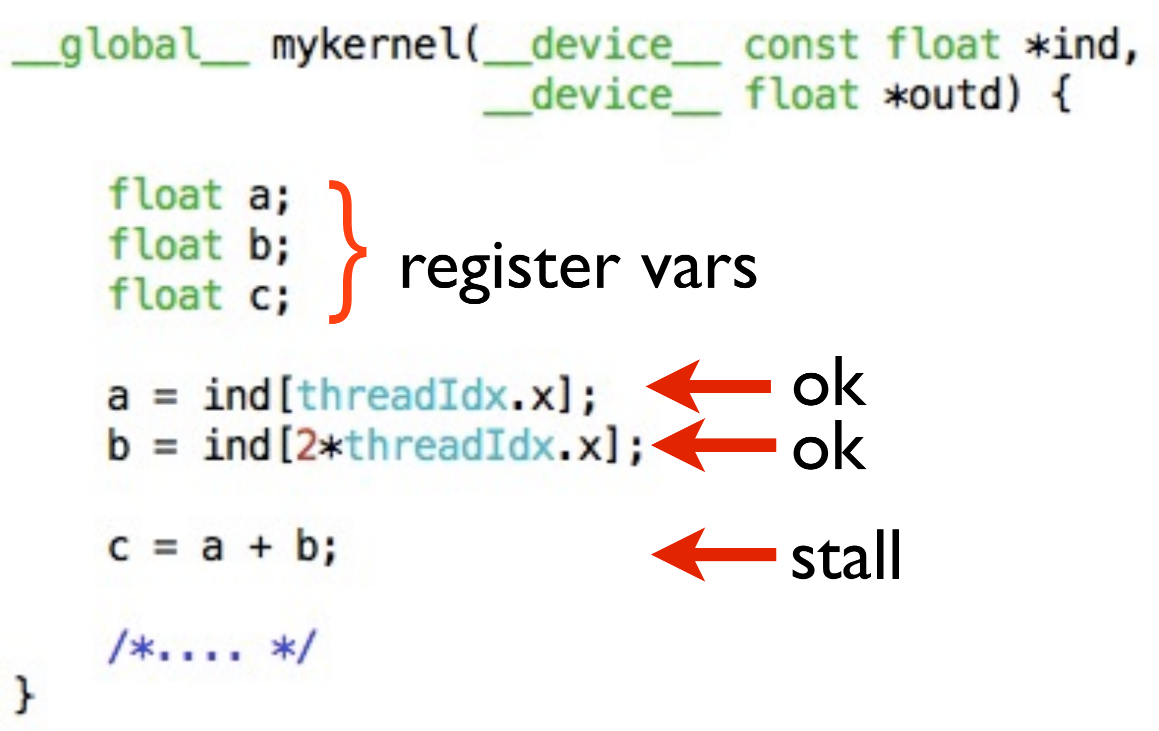
Stalling on Memory Access

- Two ways to ensure no idle SM:
  - Lots of warps (=blocks*threads/32); hide latency with other threads.
  - Little or no stalling on memory access; hide latency within threads.
- Sometimes work to counter purposes! Must experiment to see what works best for your algorithm.



Stalling happens on *use*.

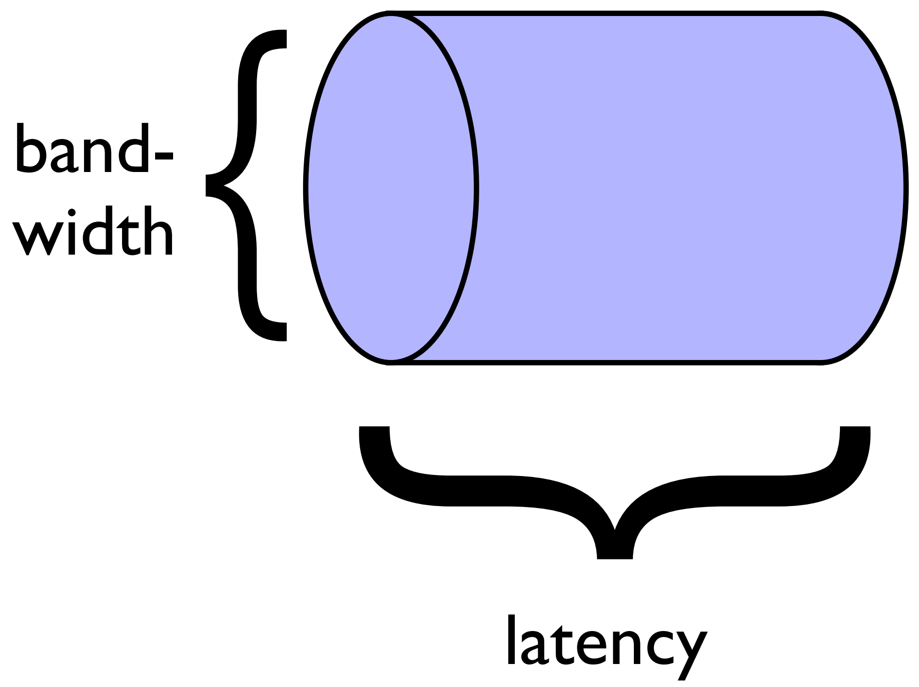
- Kernel does not stall on loading data
- Stalls when data not yet ready needs to be used
- Can “preload” data that you will need at beginning of kernel
- Hide latency by doing as much work as possible before need bulk of data.



```
__global__ mykernel(__device__ const float *ind,
                    __device__ float *outd) {

   float a;
   float b;
   float c;

   a = ind[threadIdx.x];
   b = ind[2*threadIdx.x];

   c = a + b;

   /*.... */
}
```

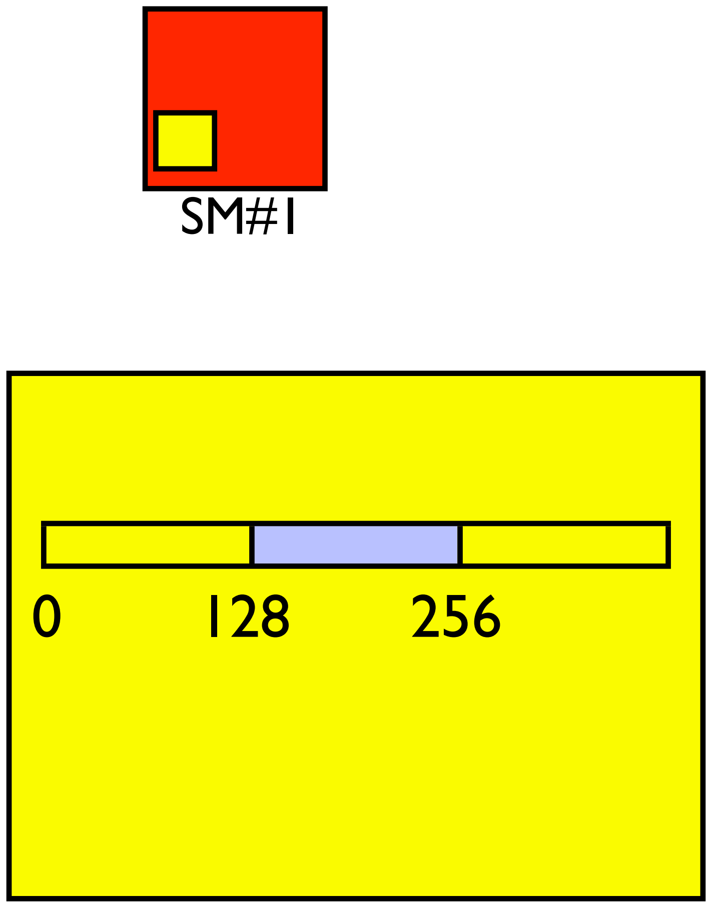
Keep memory accesses going

- Make maximum use of memory bandwidth hardware provides
- To fully use a pipe, must have bandwidth x latency memory accesses 'in flight'.
- Little's Law, Queueing theory - http://en.wikipedia.org/wiki/Little%27s_law



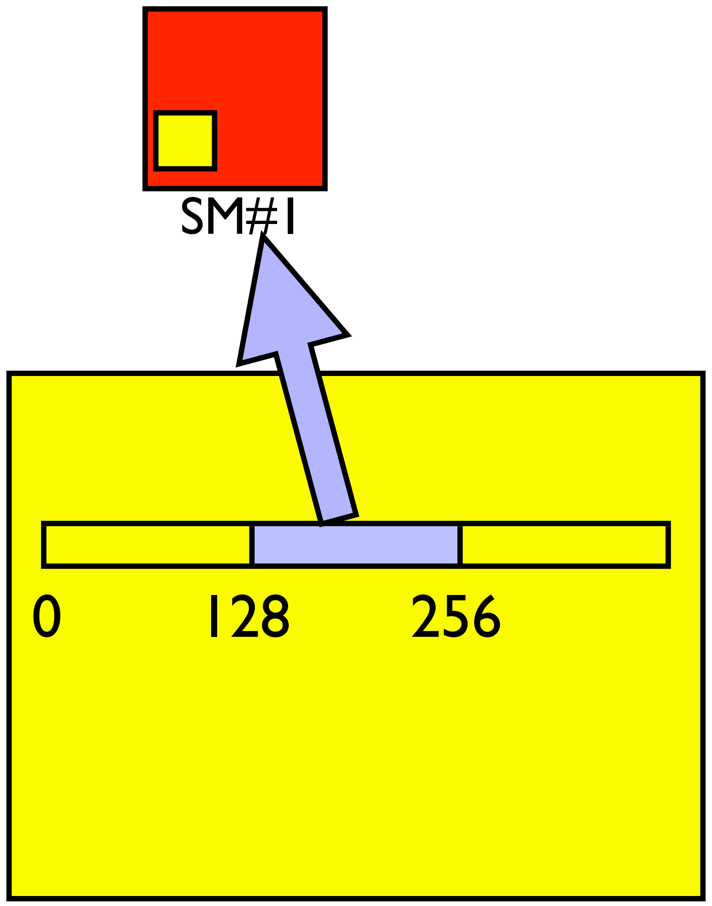
Coalesced Memory Access

- Global memory is slow
- Get as much out of it per access as possible
- HW reads 128 byte lines from global memory (Fermi: can turn off caching and read 4x 32byte segments)
- Want to make the most of this



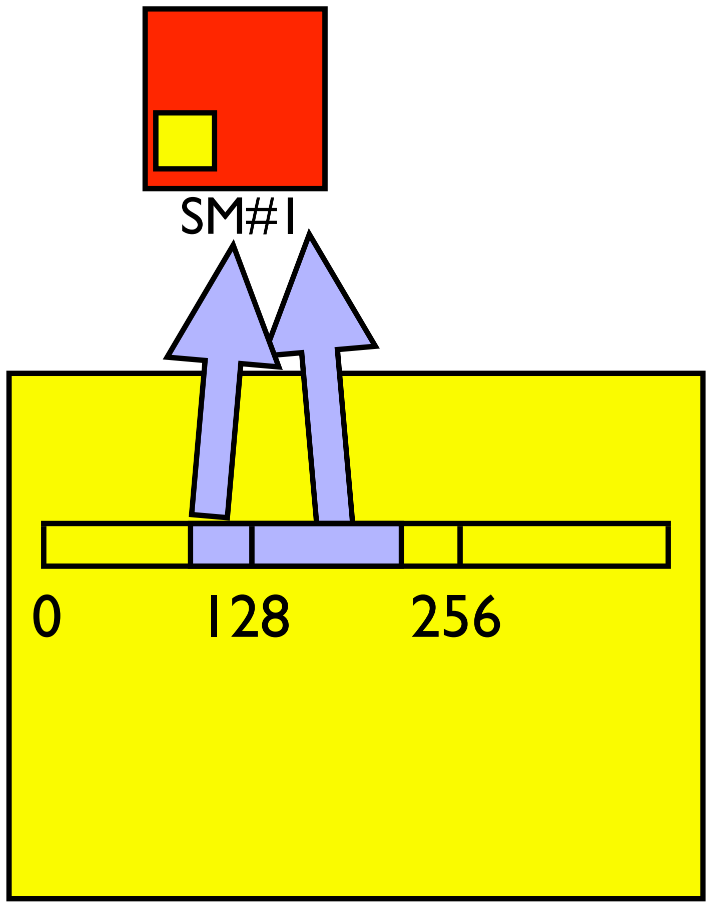
Coalesced Memory Access

- Corresponds to 4B for each thread in a warp
- If each thread in warp reads consecutive float, aligned w/ boundary, can be coalesced into 1 read: high bandwidth
- Warp can continue after 1 global read cycle



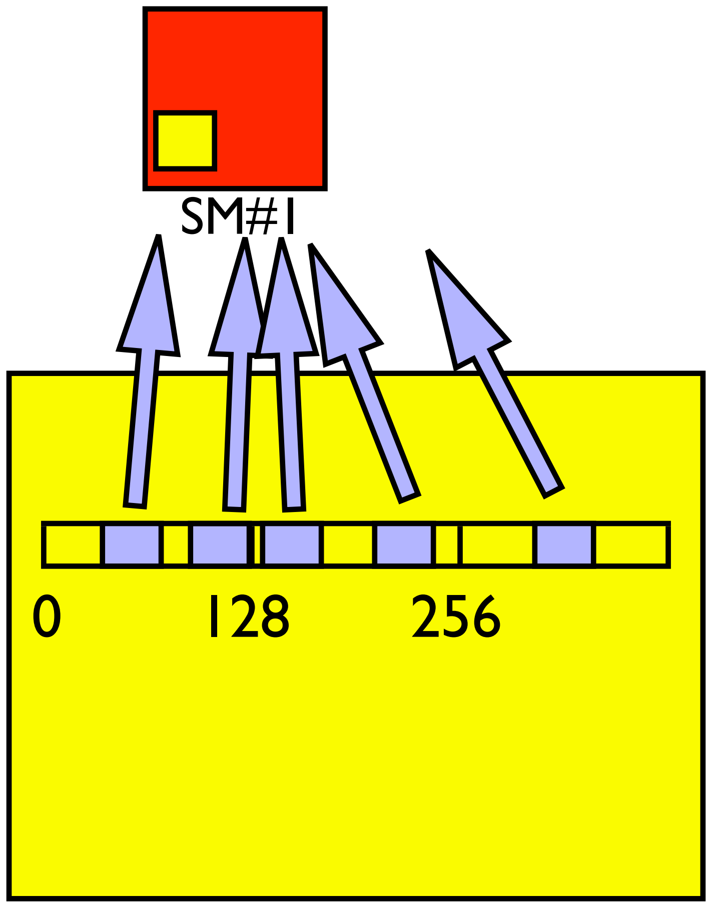
Coalesced Memory Access

- If each thread in warp reads consecutive float, but offset, can be coalesced into 2 read: reduced bandwidth
- Warp can continue after 2 global read cycle (and 128B of bandwidth wasted)



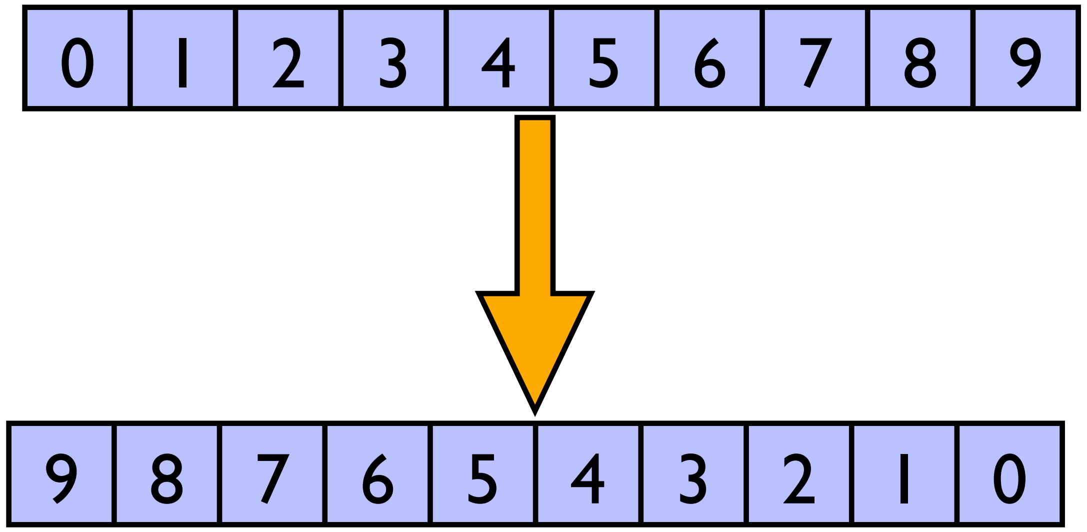
Coalesced Memory Access

- Random access is a nightmare
- Can potentially take 32 times as long, wasting 97% of available global memory bandwidth



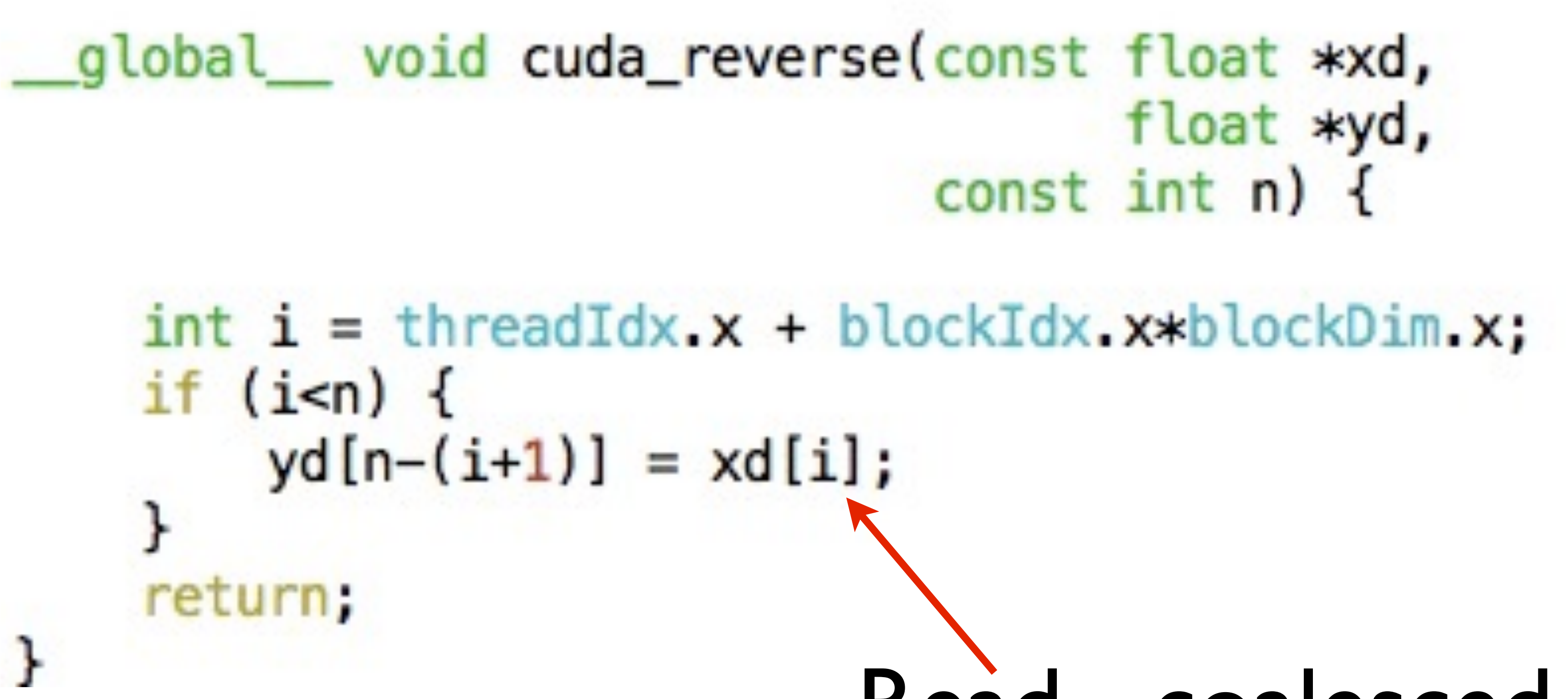
List reversal

- Imagine having to reverse a list
- (Sounds dumb, but matrix transpose, partial pivoting, various graph algorithms require data reordering)
- Obvious way to do this, particularly on older (pre cc 1.2) hardware, doesn't work well:



CITA|ICAT

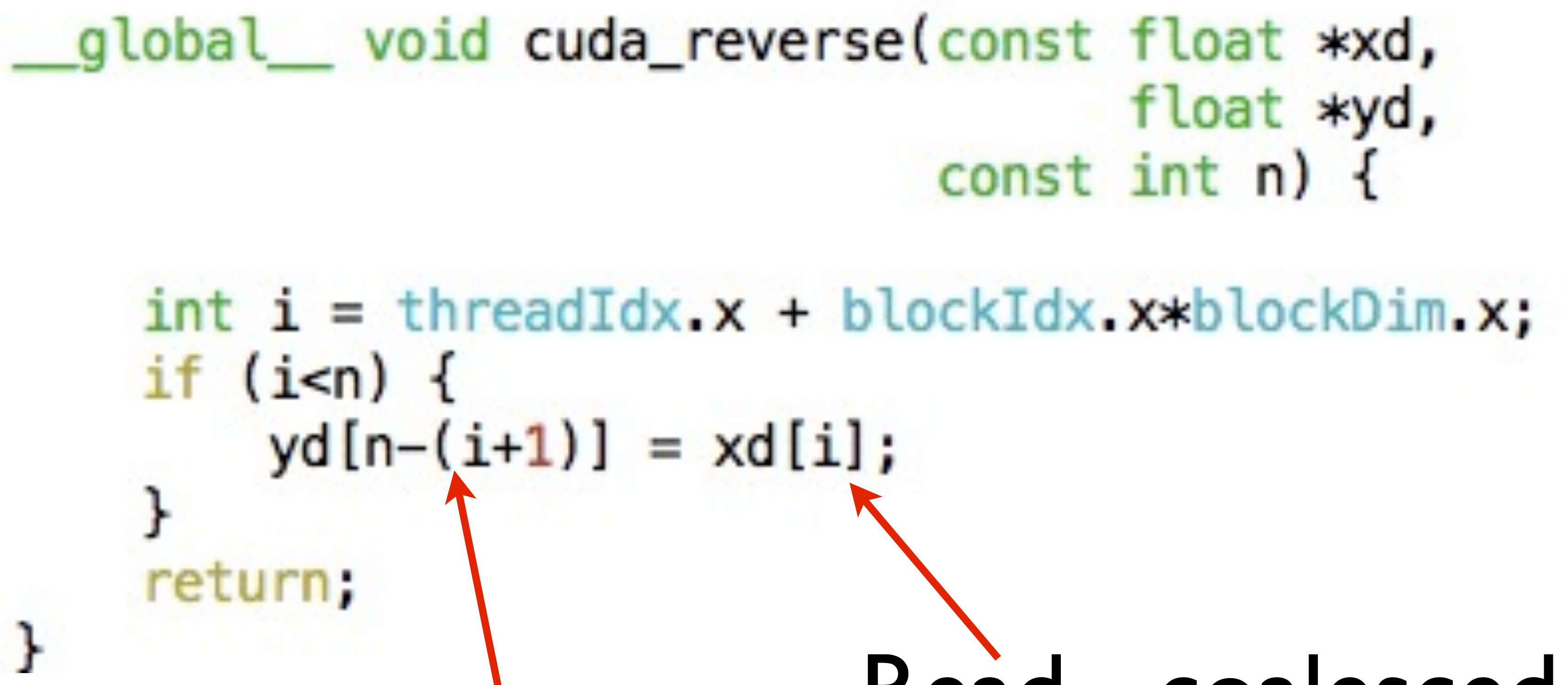
List reversal



```
__global__ void cuda_reverse(const float *xd,
                                   float *yd,
                             const int n) {

    int i = threadIdx.x + blockIdx.x*blockDim.x;
    if (i<n) {
        yd[n-(i+1)] = xd[i];
    }
    return;
}
```

Read - coalesced

List reversal

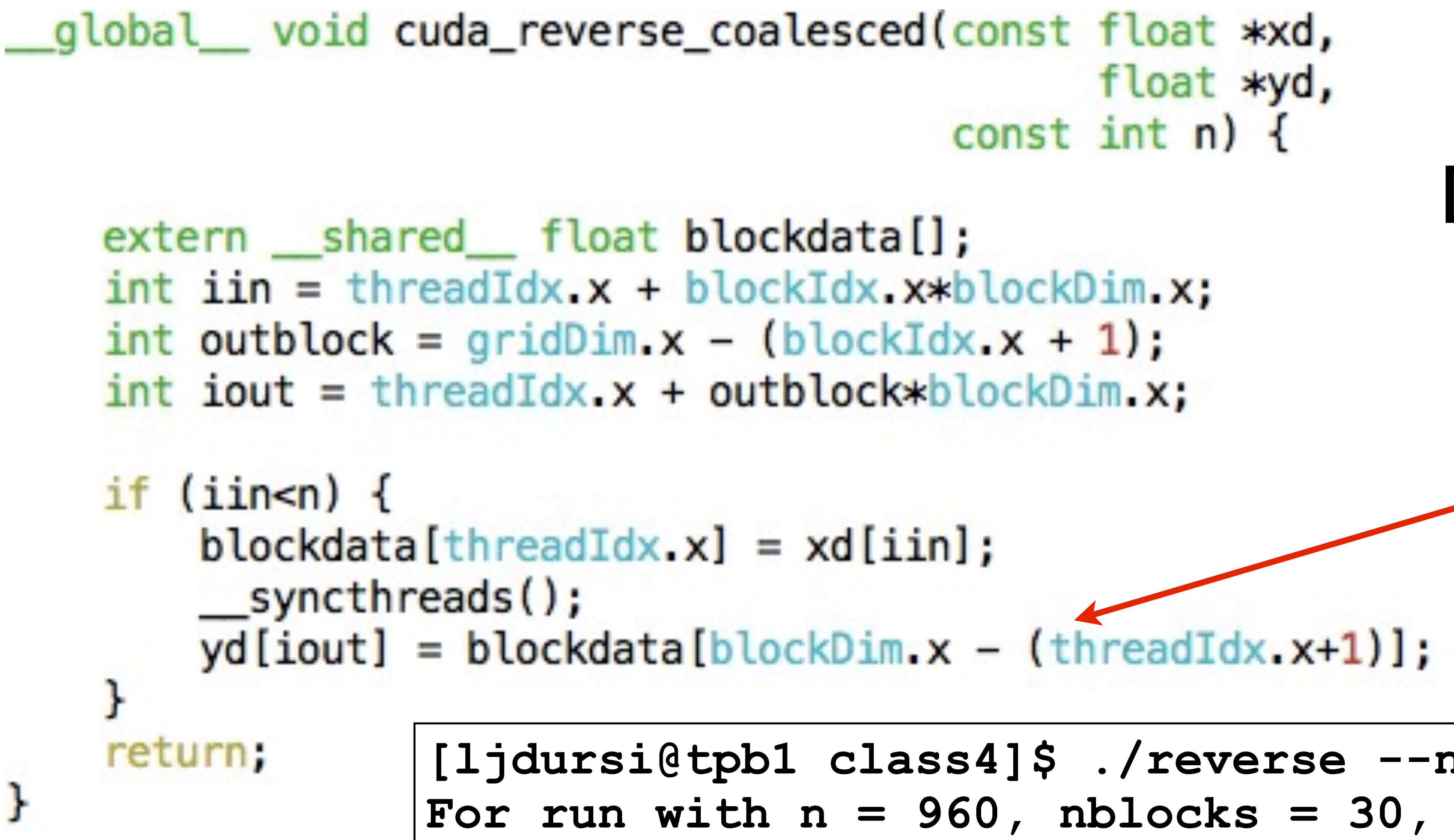


```
__global__ void cuda_reverse(const float *xd,
                                   float *yd,
                             const int n) {

    int i = threadIdx.x + blockIdx.x*blockDim.x;
    if (i<n) {
        yd[n-(i+1)] = xd[i];
    }
    return;
}
```

Read - coalesced

Write - reversed - possibly noncoalesced

List reversal



```
__global__ void cuda_reverse_coalesced(const float *xd,
                                       float *yd,
                                 const int n) {

    extern __shared__ float blockdata[];
    int iin = threadIdx.x + blockIdx.x*blockDim.x;
    int outblock = gridDim.x - (blockIdx.x + 1);
    int iout = threadIdx.x + outblock*blockDim.x;

    if (iin<n) {
        blockdata[threadIdx.x] = xd[iin];
        __syncthreads();
        yd[iout] = blockdata[blockDim.x - (threadIdx.x+1)];
    }
    return;
}
```

Do permutation in **shared** memory

```
[ljdursi@tpb1 class4]$ ./reverse --nvals=960 --nblocks=
For run with n = 960, nblocks = 30, blocksize = 32,
iters=1,
CPU time  = 0.002 millisec.
GPU time  = 0.101 millisec, diff = 0.000000.
GPU2 time = 0.059 millisec, diff = 0.000000.
```

CITA|ICAT

HW2

```
__global__ void cuda_smoothimage(uint8_t *data,
                                 uint8_t *smootheddata,
                                 const int nhalos, const int rows, const int cols) {

    const double sig = 2./9.*(nhalos*nhalos);
    const double pi = 4.*atan(1.);
    const int rowsize=cols+2*nhalos;

    int r = threadIdx.y + blockIdx.y*blockDim.y;
    int c = threadIdx.x + blockIdx.x*blockDim.x;

    if (r<rows && c<cols) {
        r += nhalos;
        c += nhalos;
        double mean = 0.;
        for (int i=r-nhalos; i<=r+nhalos; i++) {
            for (int j=c-nhalos; j<=c+nhalos; j++) {
                double d  = 1.0*data[i*rowsize+j];
                double e = exp(-(double)((i-r)*(i-r)+(j-c)*(j-c))/sig);
                mean += d*e;
            }
        }
        smootheddata[r*rowsize + c] = (uint8_t)round(mean/(sig*pi));
    }
    return;
}
```

global memory access

HW2

```
__global__ void cuda_smoothimage(uint8_t *data,
                                 uint8_t *smootheddata,
                                 const int nhalos, const int rows, const int cols) {

    const double sig = 2./9.*(nhalos*nhalos);
    const double pi = 4.*atan(1.);
    const int rowsize=cols+2*nhalos;

    int r = threadIdx.y + blockIdx.y*blockDim.y;
    int c = threadIdx.x + blockIdx.x*blockDim.x;
```

CUDA thread blocks are organized filling in x direction first, then y, then z; (x,y,z) x is fastest moving.

Map to image: columns are fastest varying, then rows.

So this thread ordering has thread #1 accessing pixel 1, thread #2 accessing pixel 2, etc... coallesced.

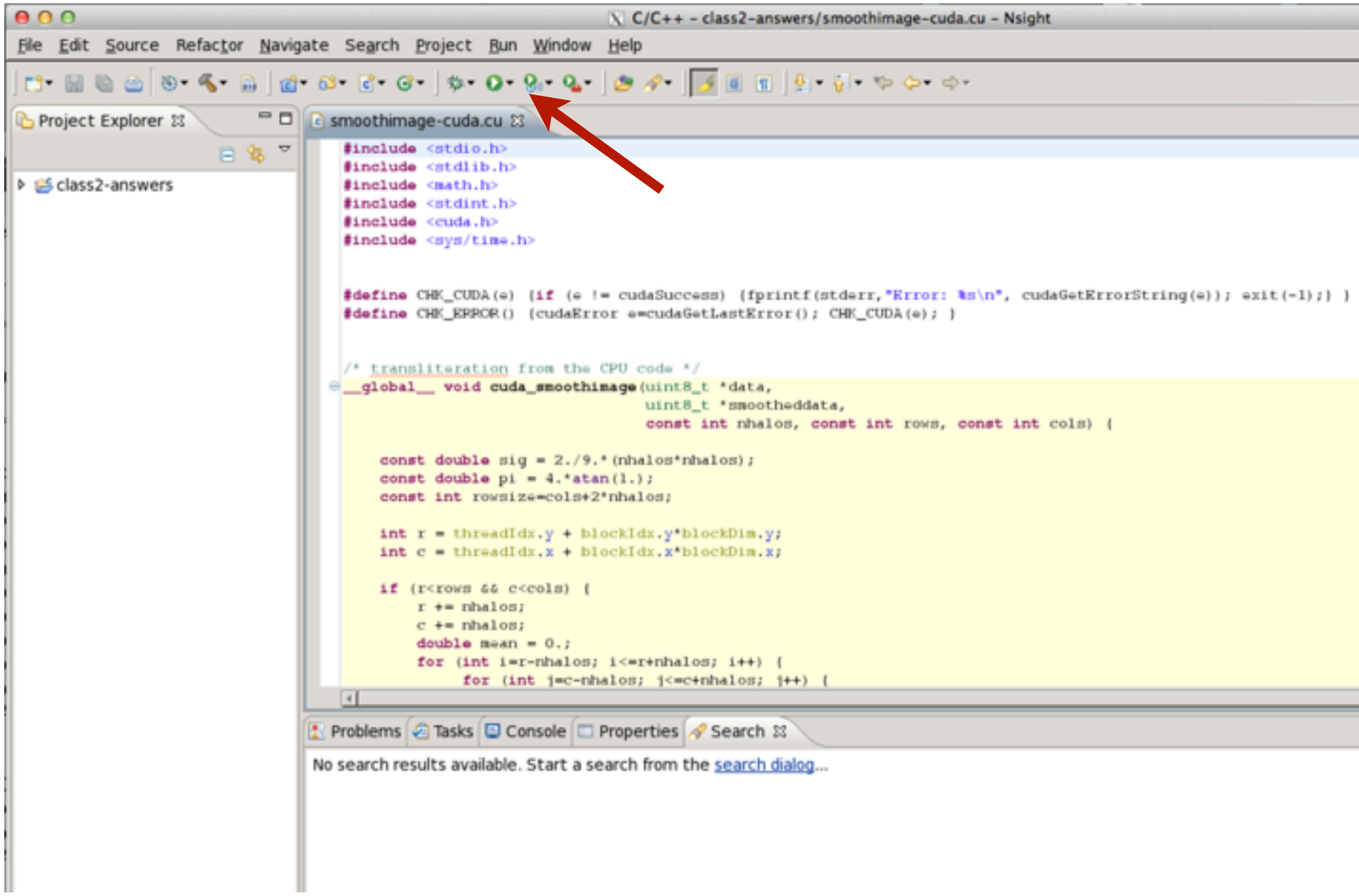
```
    }
    return;
}
```

```
__global__ void cuda_smoothimage_uncoallesced(uint8_t *data,
                                 uint8_t *smootheddata,
                                 const int nhalos, const int rows, const int cols) {

    const double sig = 2./9.*(nhalos*nhalos);
    const double pi = 4.*atan(1.);
    const int rowsize=cols+2*nhalos;

    int c = threadIdx.y + blockIdx.y*blockDim.y;
    int r = threadIdx.x + blockIdx.x*blockDim.x;

    if (r<rows && c<cols) {
        r += nhalos;
        c += nhalos;
        double mean = 0.;
        for (int i=r-nhalos; i<=r+nhalos; i++) {
            for (int j=c-nhalos; j<=c+nhalos; j++) {
                double d  = 1.0*data[i*rowsize+j];
                double e = exp(-(double)((i-r)*(i-r)+(j-c)*(j-c))/sig);
                mean += d*e;
            }
        }
        smootheddata[r*rowsize + c] = (uint8_t)round(mean/(sig*pi));
    }
    return;
}
```

(+flip row, col in grid)

NSight, Eclipse edition

- For Mac, Linux in CUDA 5.0
- (NSight for Studio for win earlier)
- type "nsight", put into IDE with debugger, profiler, etc



C/C++ - class2-answers/smoothimage-cuda.cu - Nsight
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
class2-answers
smoothimage-cuda.cu
Problems Tasks Console Properties Search

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <stdint.h>
#include <cuda.h>
#include <sys/time.h>


#define CHK_CUDA(e) {if (e != cudaSuccess) {fprintf(stderr,"Error: %s\n", cudaGetErrorString(e)); exit(-1);}}
#define CHK_ERROR() {cudaError e=cudaGetLastError(); CHK_CUDA(e); }


/* transliteration from the CPU code */
__global__ void cuda_smoothimage(uint8_t *data,
                                 uint8_t *smootheddata,
                                 const int nhalos, const int rows, const int cols) {

    const double sig = 2./9.*(nhalos*nhalos);
    const double pi = 4.*atan(1.);
    const int rowsize=cols+2*nhalos;

    int r = threadIdx.y + blockIdx.y*blockDim.y;
    int c = threadIdx.x + blockIdx.x*blockDim.x;

    if (r<rows && c<cols) {
        r += nhalos;
        c += nhalos;
        double mean = 0.;
        for (int i=r-nhalos; i<=r+nhalos; i++) {
             for (int j=c-nhalos; j<=c+nhalos; j++) {
```

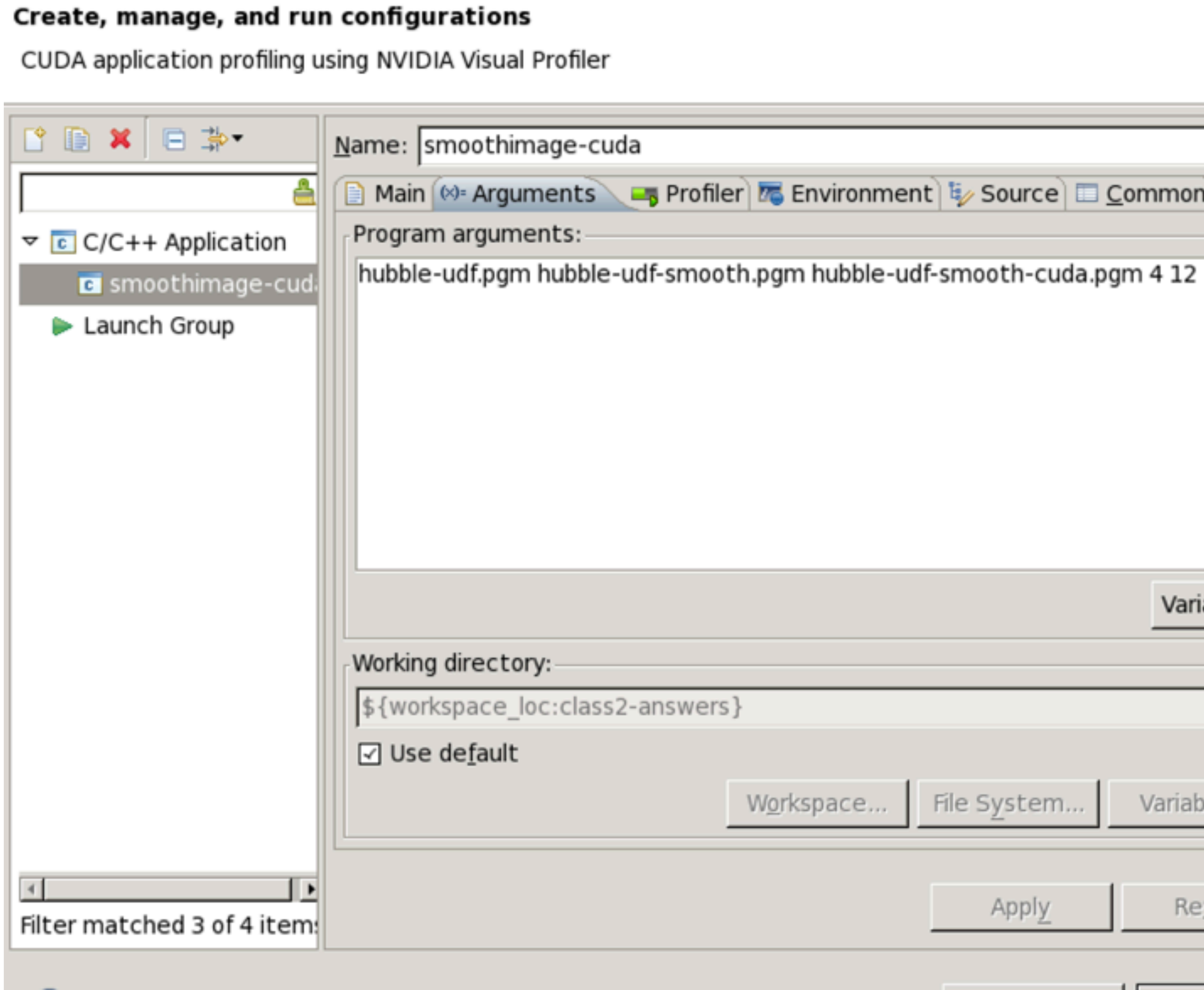
No search results available. Start a search from the search dialog...

CITA|ICAT

SciNet

Profile Configurations

- Under profile menu, Profile Configurations will let you choose the executable, arguments to profile
- Then clicking "profile" takes you into profiling perspective, does timeline.


Create, manage, and run configurations

CUDA application profiling using NVIDIA Visual Profiler

Name: smoothimage-cuda

Program arguments:

hubble-udf.pgm hubble-udf-smooth.pgm hubble-udf-smooth-cuda.pgm 4 12

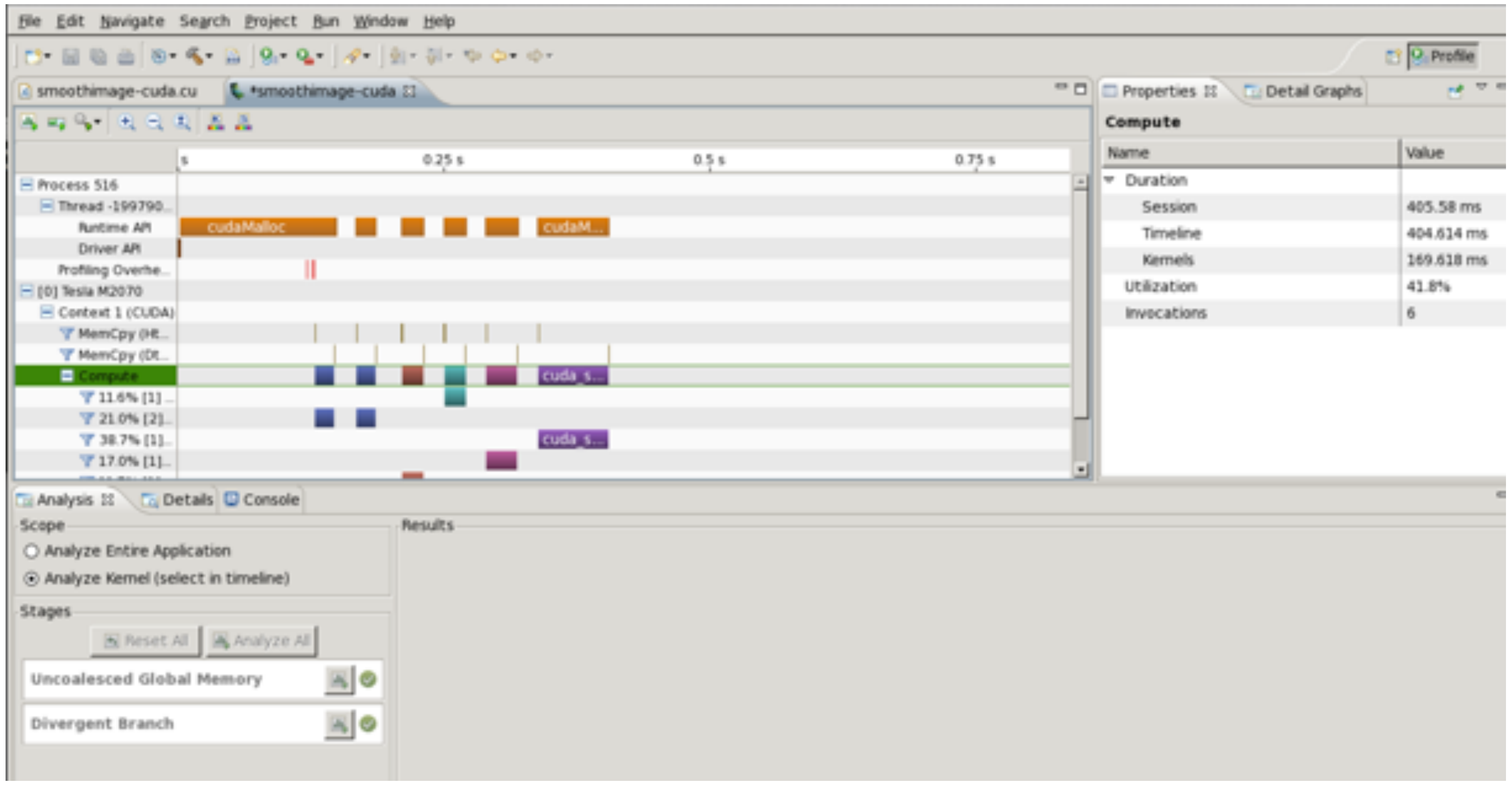
Working directory:

${workspace_loc:class2-answers}

☑ Use default

CITA|ICAT

Profile Configurations



Compute

Name	Value
Duration	
Session	405.58 ms
Timeline	404.614 ms
Kernels	169.618 ms
Utilization	41.8%
Invocations	6

Analyze Entire Application [✓ Analyze Kernel (select in timeline)]

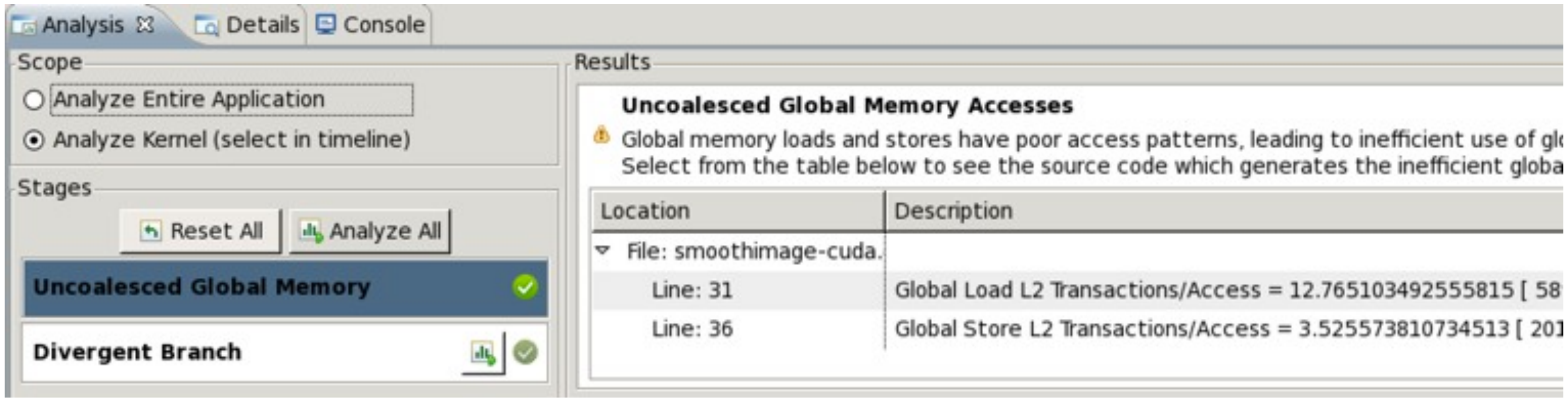
Uncoalesced Global Memory

Divergent Branch

- Initial time line gives overview of kernels duration for entire run
- "Analyze entire application" also lets you see if you're keeping multiple GPUs busy, etc.
- Analyze Kernels lets you get stats about particular kernels.

CITA|ICAT

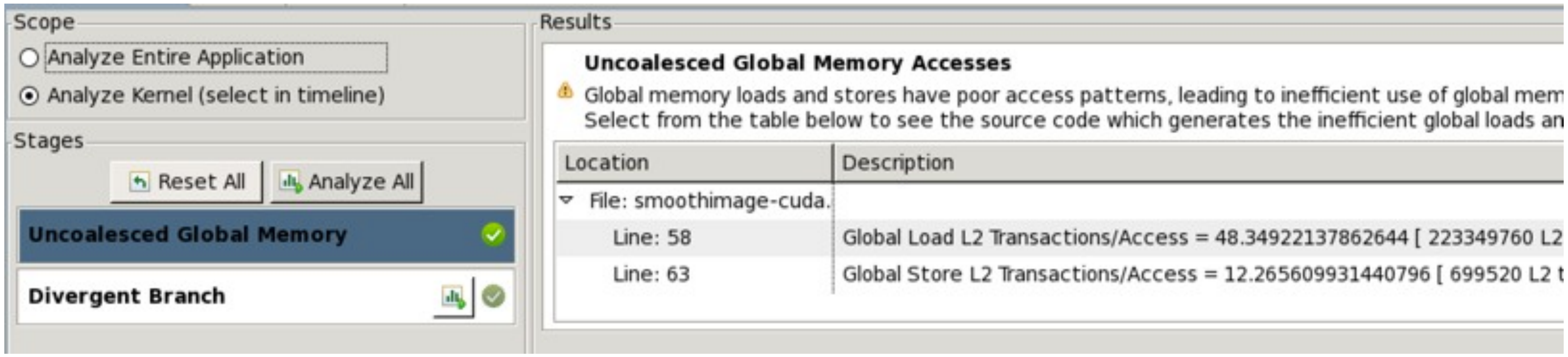
SciNet


Analysis
Details
Console
Scope
Analyze Entire Application
Analyze Kernel (select in timeline)
Stages
Reset All
Analyze All
Uncoalesced Global Memory
Divergent Branch
Results

Uncoalesced Global Memory Accesses

Global memory loads and stores have poor access patterns, leading to inefficient use of gl
Select from the table below to see the source code which generates the inefficient globa

Location	Description
File: smoothimage-cuda.	
Line: 31	Global Load L2 Transactions/Access = 12.765103492555815 [58
Line: 36	Global Store L2 Transactions/Access = 3.525573810734513 [201


Scope
Analyze Entire Application
Analyze Kernel (select in timeline)
Stages
Reset All
Analyze All
Uncoalesced Global Memory
Divergent Branch
Results

Uncoalesced Global Memory Accesses

Global memory loads and stores have poor access patterns, leading to inefficient use of global mem
Select from the table below to see the source code which generates the inefficient global loads an

Location	Description
File: smoothimage-cuda.	
Line: 58	Global Load L2 Transactions/Access = 48.34922137862644 [223349760 L2
Line: 63	Global Store L2 Transactions/Access = 12.265609931440796 [699520 L2 t

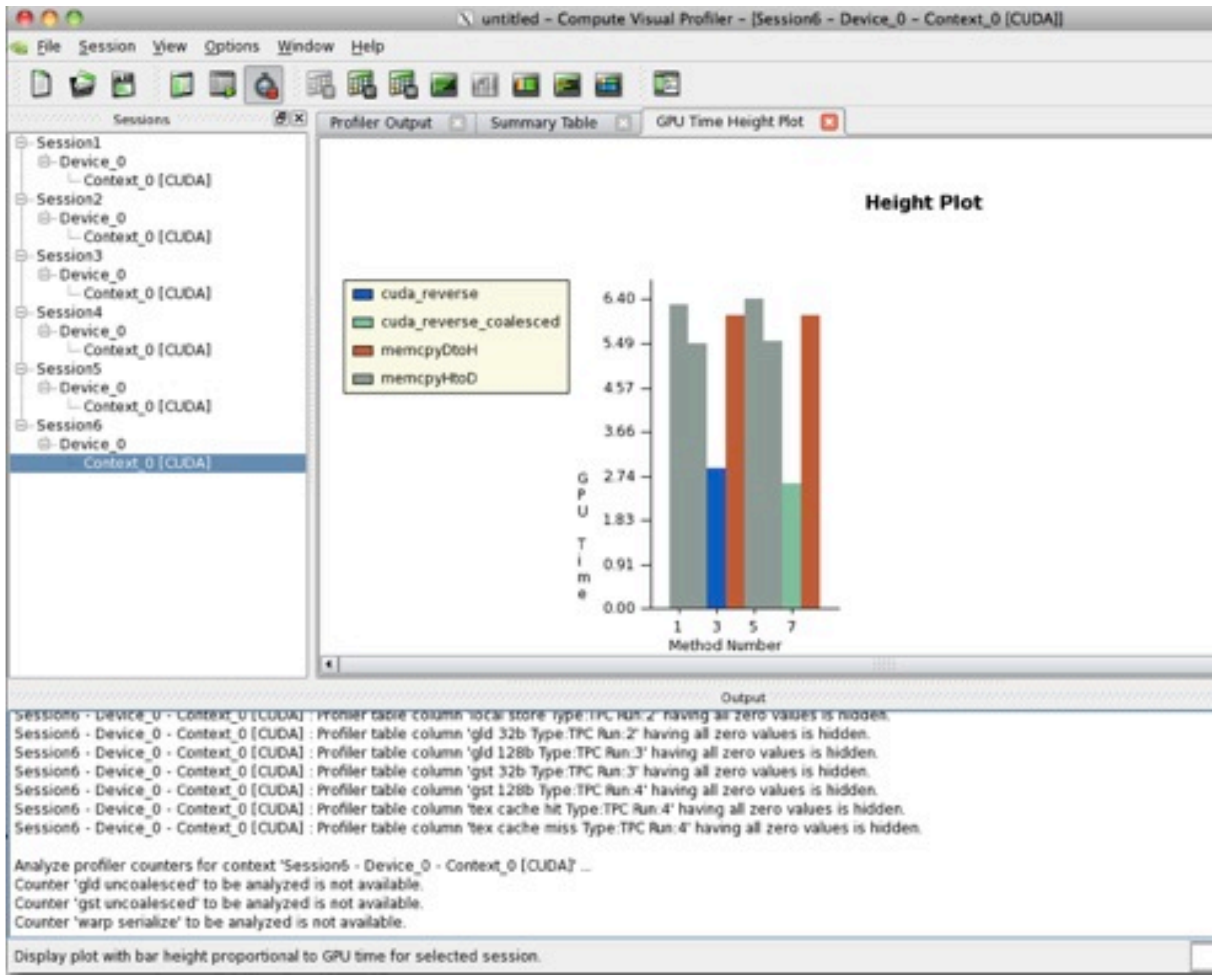
- Have to do for each kernel under consideration
- Profiler may have to run multiple times each to get all the data
- Uncoallesced one has (even) worse memory access: more transactions (4x)

CITA|ICAT

SciNet

Visual Profiler

- Cuda/OpenCL profiler comes with NVidia SDK 3.2, 4.0
- run with `computeprof`
- From there, you can run an application and look at timings

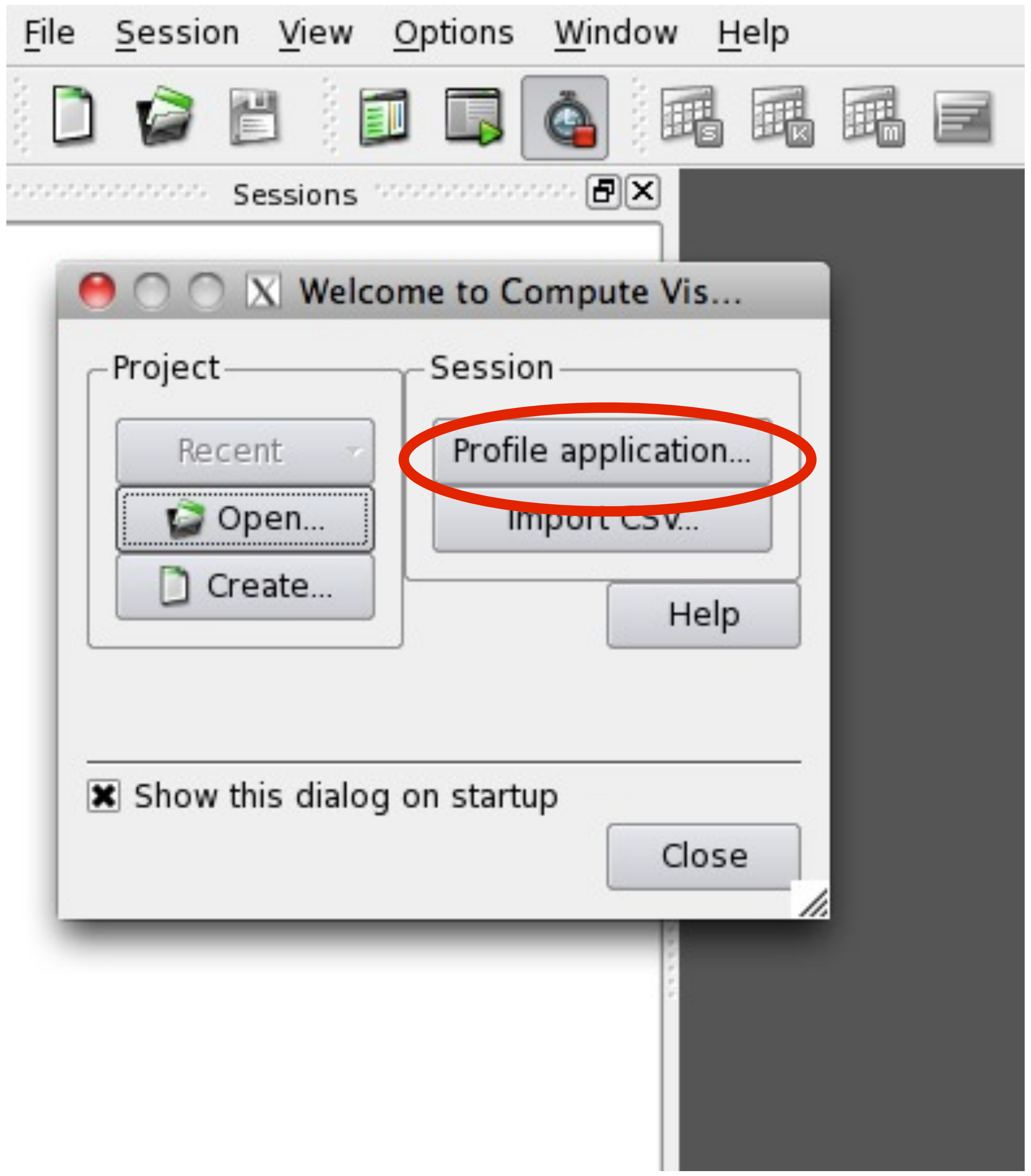


```
Session6 - Device_0 - Context_0 [CUDA] : Profiler table column 'local store Type:TPC Run:2' having all zero values is hidden.
Session6 - Device_0 - Context_0 [CUDA] : Profiler table column 'gld 32b Type:TPC Run:2' having all zero values is hidden.
Session6 - Device_0 - Context_0 [CUDA] : Profiler table column 'gld 128b Type:TPC Run:3' having all zero values is hidden.
Session6 - Device_0 - Context_0 [CUDA] : Profiler table column 'gst 32b Type:TPC Run:3' having all zero values is hidden.
Session6 - Device_0 - Context_0 [CUDA] : Profiler table column 'gst 128b Type:TPC Run:4' having all zero values is hidden.
Session6 - Device_0 - Context_0 [CUDA] : Profiler table column 'tex cache hit Type:TPC Run:4' having all zero values is hidden.
Session6 - Device_0 - Context_0 [CUDA] : Profiler table column 'tex cache miss Type:TPC Run:4' having all zero values is hidden.

Analyze profiler counters for context 'Session6 - Device_0 - Context_0 [CUDA]' ...
Counter 'gld uncoalesced' to be analyzed is not available.
Counter 'gst uncoalesced' to be analyzed is not available.
Counter 'warp serialize' to be analyzed is not available.
```

Display plot with bar height proportional to GPU time for selected session.

Visual Profiler

- Click 'Profile application' to begin getting data,

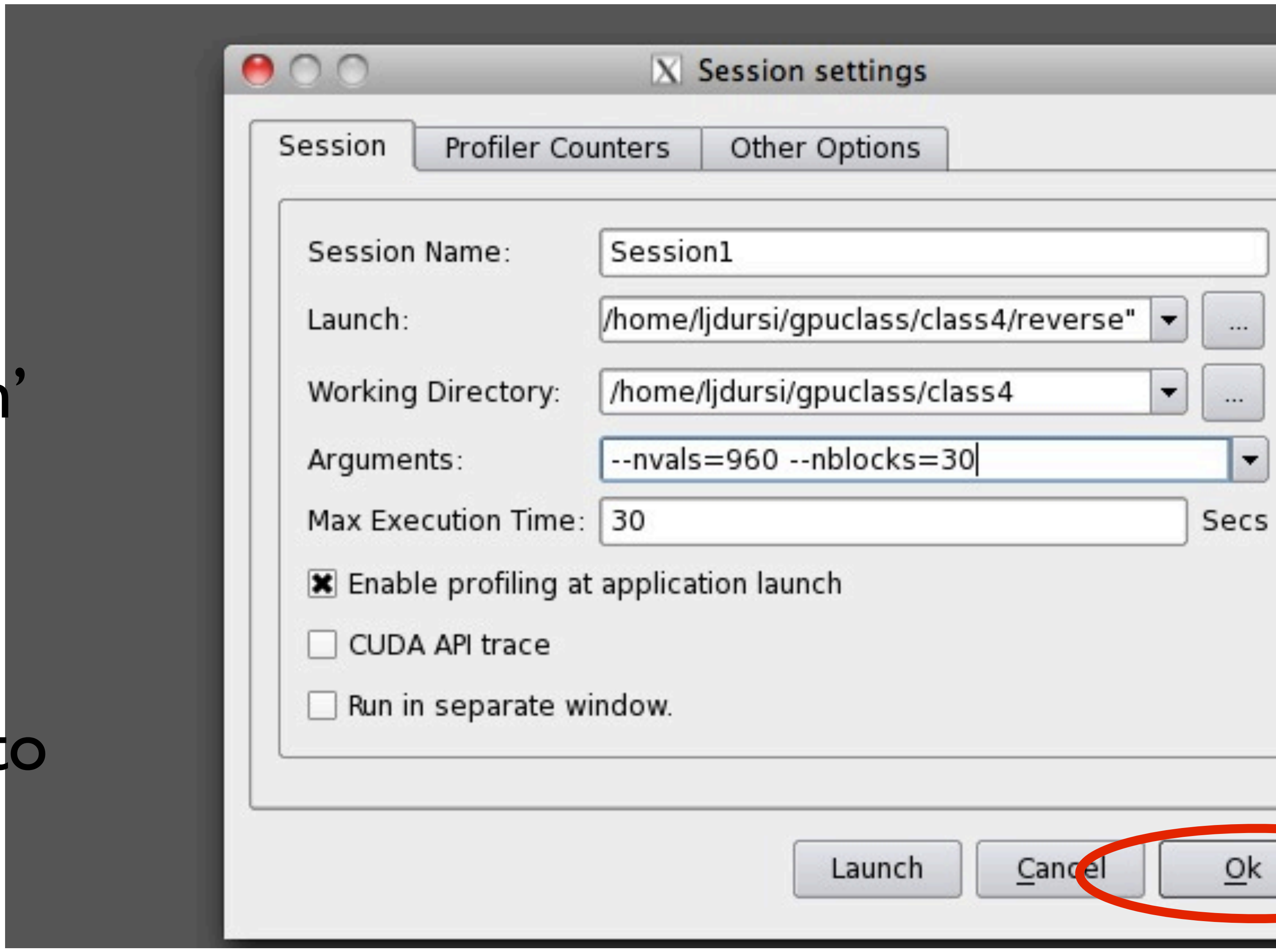


CITA|ICAT

SciNet

Visual Profiler

- Click 'Profile application' to begin getting data,
- Enter directory, executable, and arguments of program to profile,


Session settings

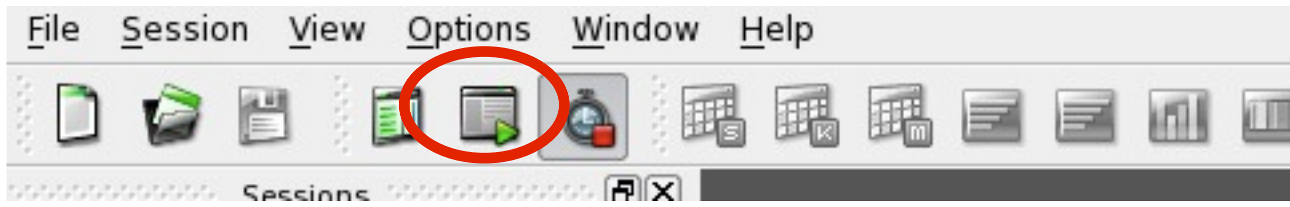
Session Name: Session1
Launch: /home/ljdursi/gpuclass/class4/reverse"
Working Directory: /home/ljdursi/gpuclass/class4
Arguments: --nvals=960 --nblocks=30
Max Execution Time: 30 Secs
[✓] Enable profiling at application launch
[ ] CUDA API trace
[ ] Run in separate window.

CITA|ICAT

SciNet

Visual Profiler

- Click 'Profile application' to begin getting data,
- Enter directory, executable, and arguments of program to profile,
- and then run the program. Program runs several times to get all counter information.

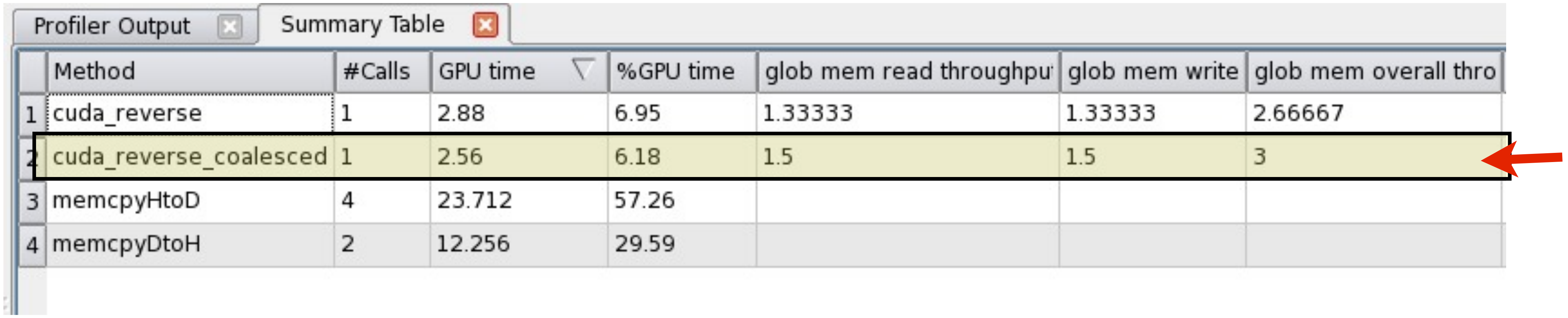


CITA|ICAT

SciNet

Visual Profiler

- Summary table shows lots of good stuff
- Here we see overall *kernel* time is about 12% faster, presumably because of roughly ~12% better global memory throughput.



Profiler Output | Summary Table

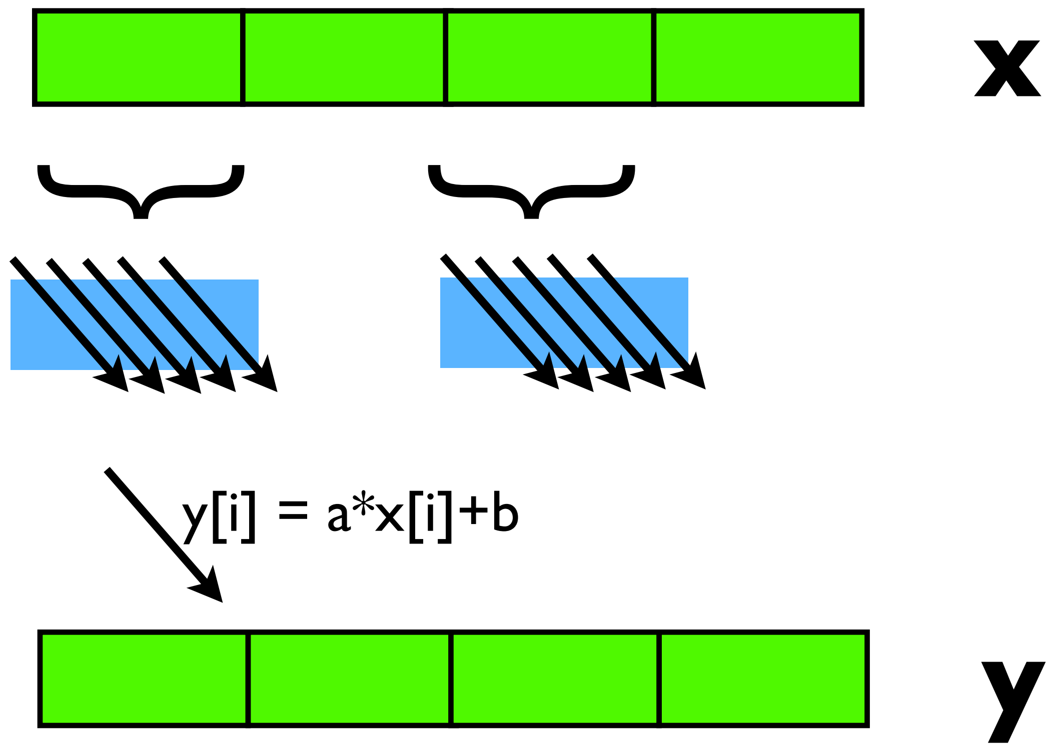
	Method	#Calls	GPU time	%GPU time	glob mem read throughpu	glob mem write	glob mem overall thro
1	cuda_reverse	1	2.88	6.95	1.33333	1.33333	2.66667
2	cuda_reverse_coalesced	1	2.56	6.18	1.5	1.5	3
3	memcpyHtoD	4	23.712	57.26			
4	memcpyDtoH	2	12.256	29.59			

CITA|ICAT

SciNet

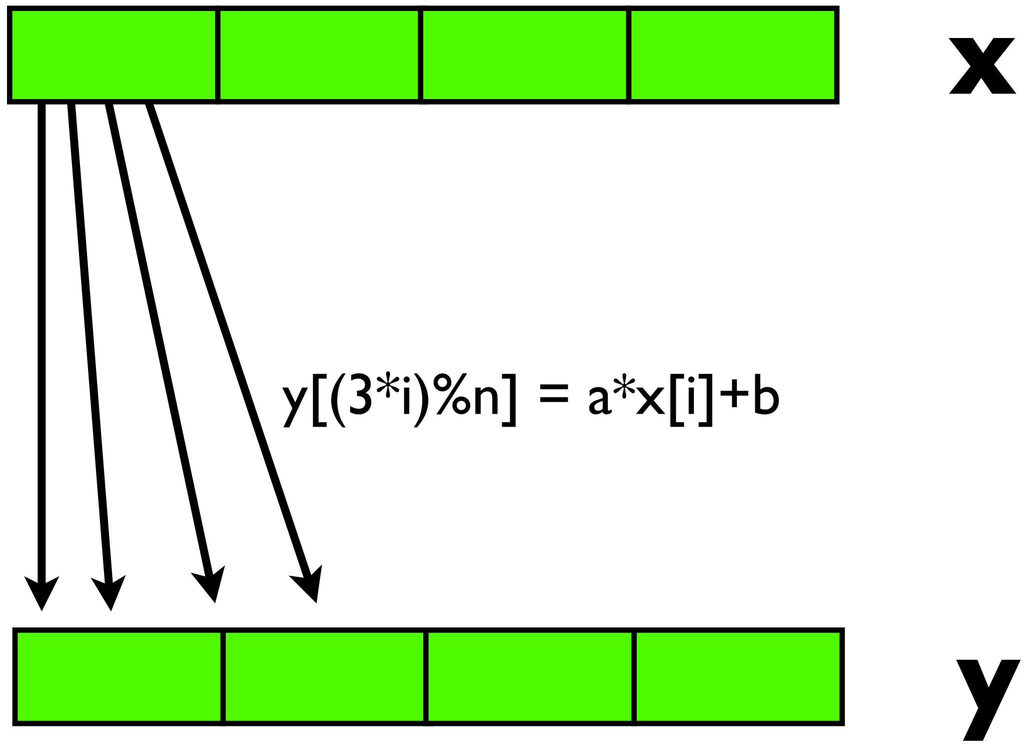
Another Example: Multi-block y=ax+b

- Break input, output vectors into blocks
- Within each block, thread index specifies which item to work on
- Each thread does one update, puts results in y[i]



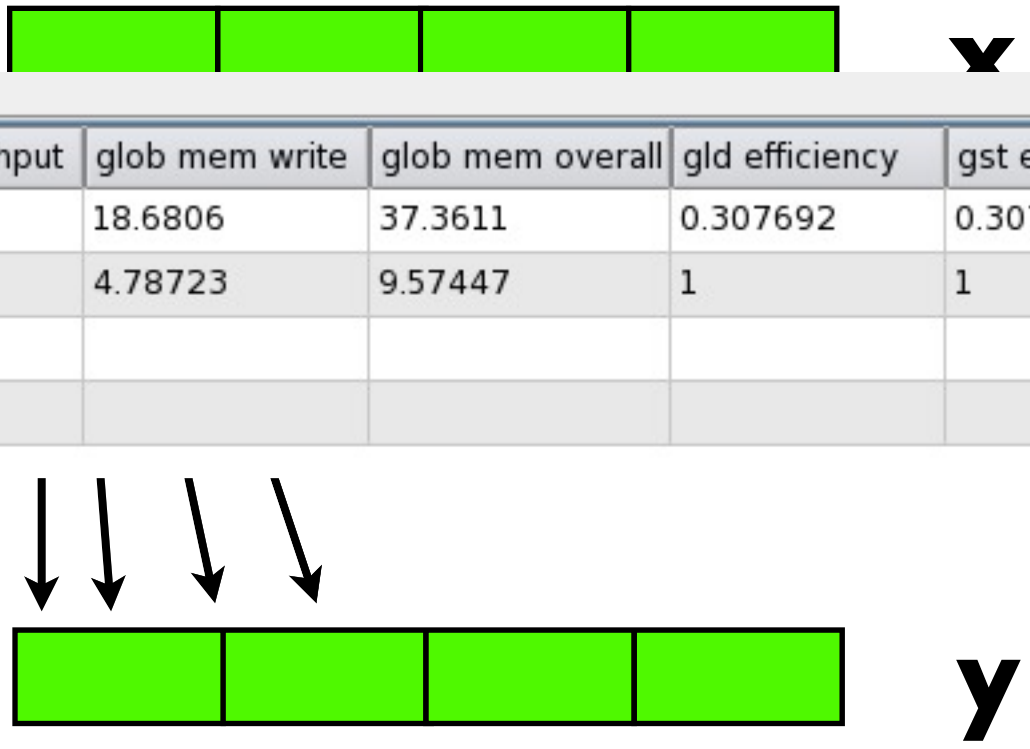
Another Example: Multi-block y=ax+b

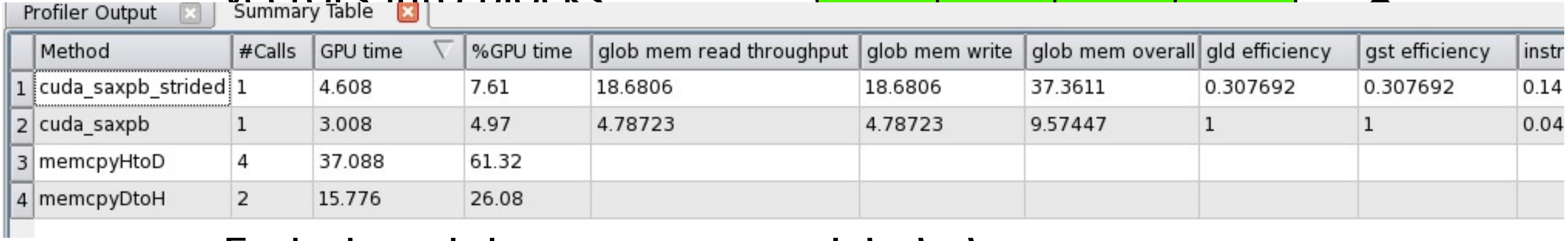
- Break input, output vectors into blocks
- Within each block, thread index specifies which item to work on
- Each thread does one update, puts results in y[i]
- But now with a stride:
- Can coalesce reads, writes, but not both.



Another Example: Multi-block y=ax+b

- Break input, output vectors into blocks





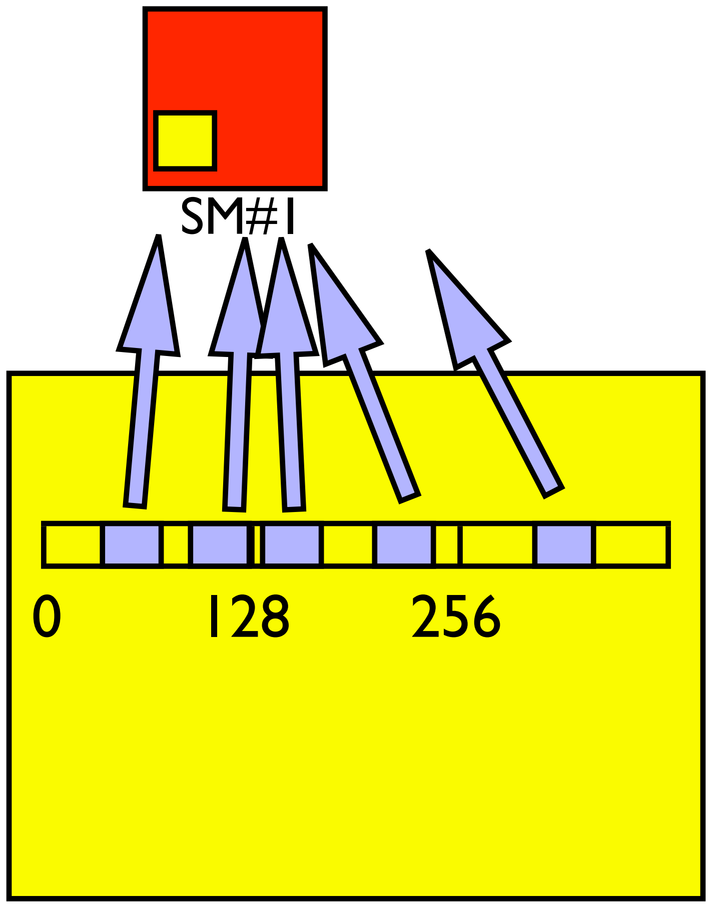
Profiler Output | Summary Table

	Method	#Calls	GPU time	%GPU time	glob mem read throughput	glob mem write	glob mem overall	gld efficiency	gst efficiency	instr
1	cuda_saxpb_strided	1	4.608	7.61	18.6806	18.6806	37.3611	0.307692	0.307692	0.14
2	cuda_saxpb	1	3.008	4.97	4.78723	4.78723	9.57447	1	1	0.04
3	memcpyHtoD	4	37.088	61.32						
4	memcpyDtoH	2	15.776	26.08						

- Each thread does one update, puts results in y[i]
- But now with a stride:
- Can coalesce reads, writes, but not both.

Coalesced Memory Access

- Rewriting algorithm to ensure coalesced memory access probably most important optimization.
- Try to rearrange data before transfer to device to be in order needed;
- Reorder in shared mem if necessary.

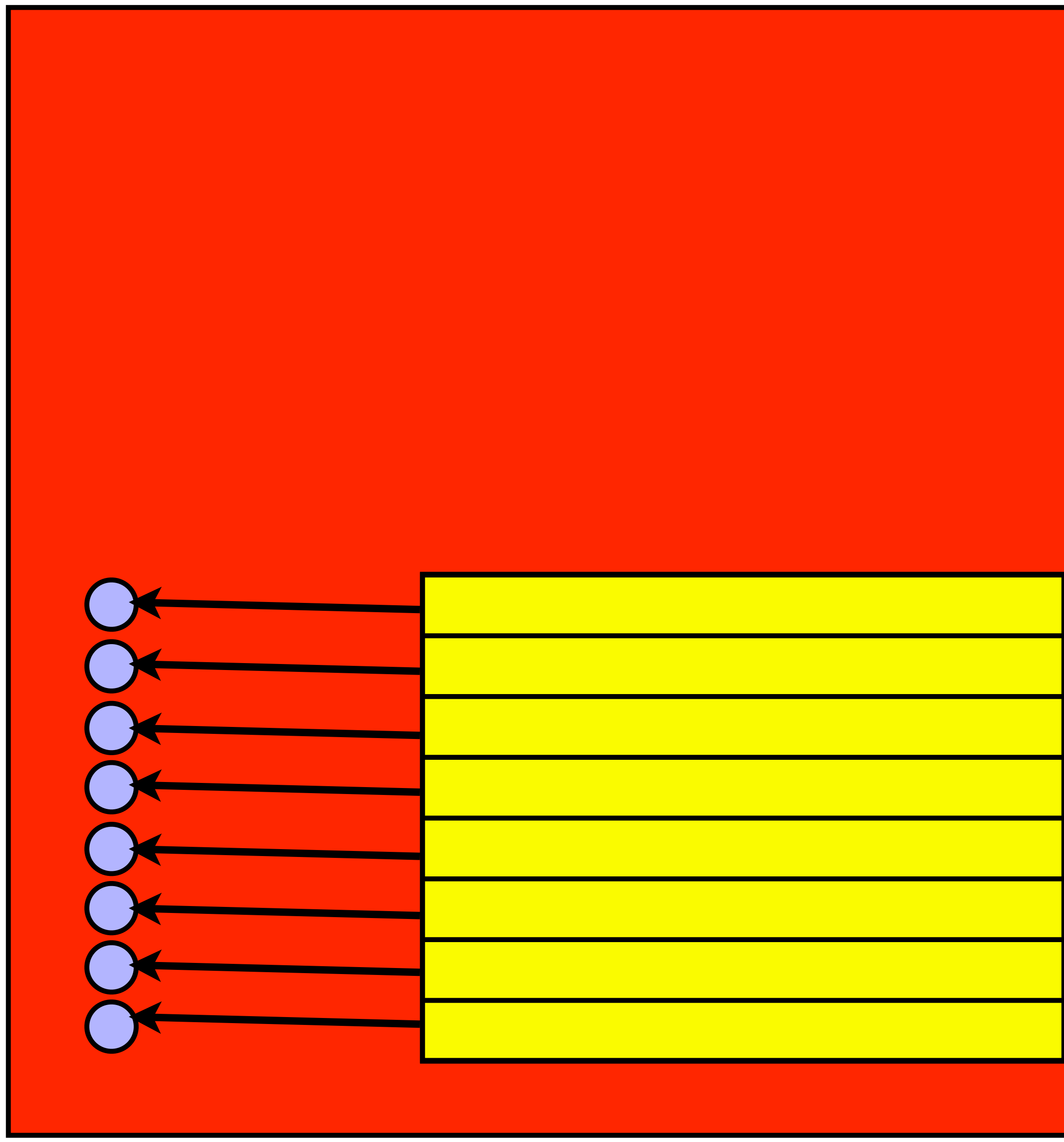


CITA|ICAT

SciNet

Shared Memory Bank Conflicts

- Each thread in warp accesses different bank: no problem.

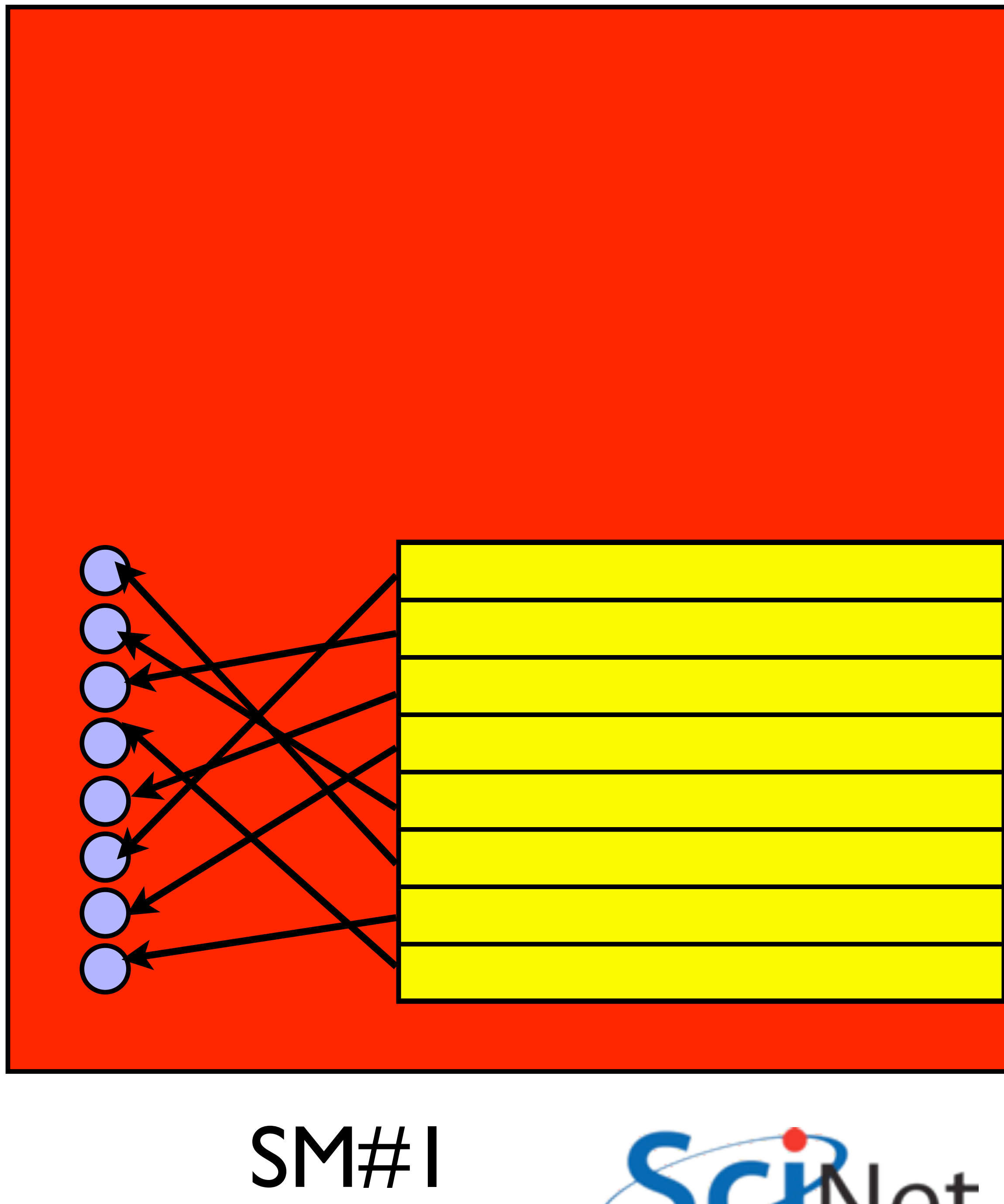


SM#1

CITA|ICAT

Shared Memory Bank Conflicts

- Each thread in warp accesses different bank: no problem.

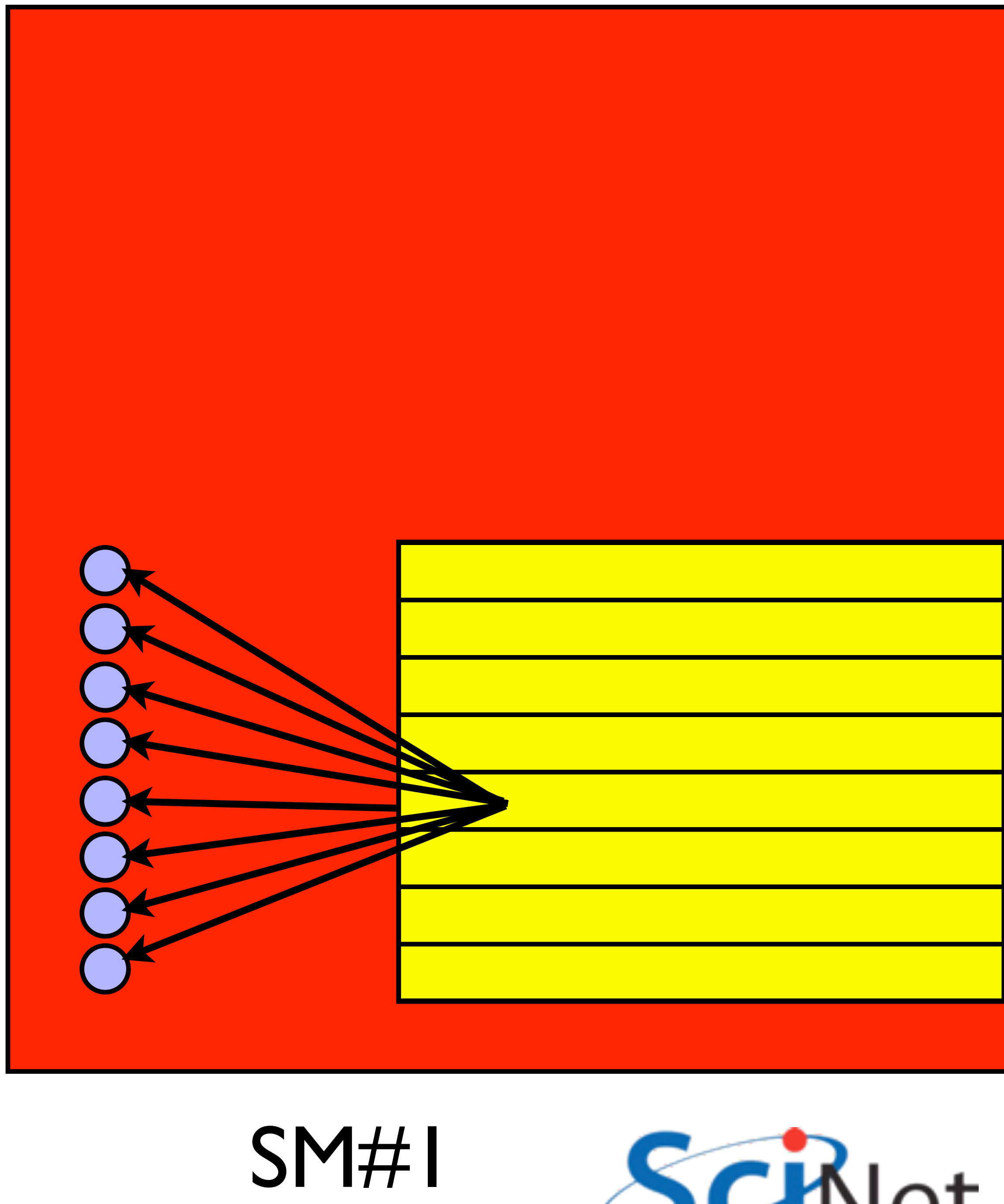


CITA|ICAT

SciNet

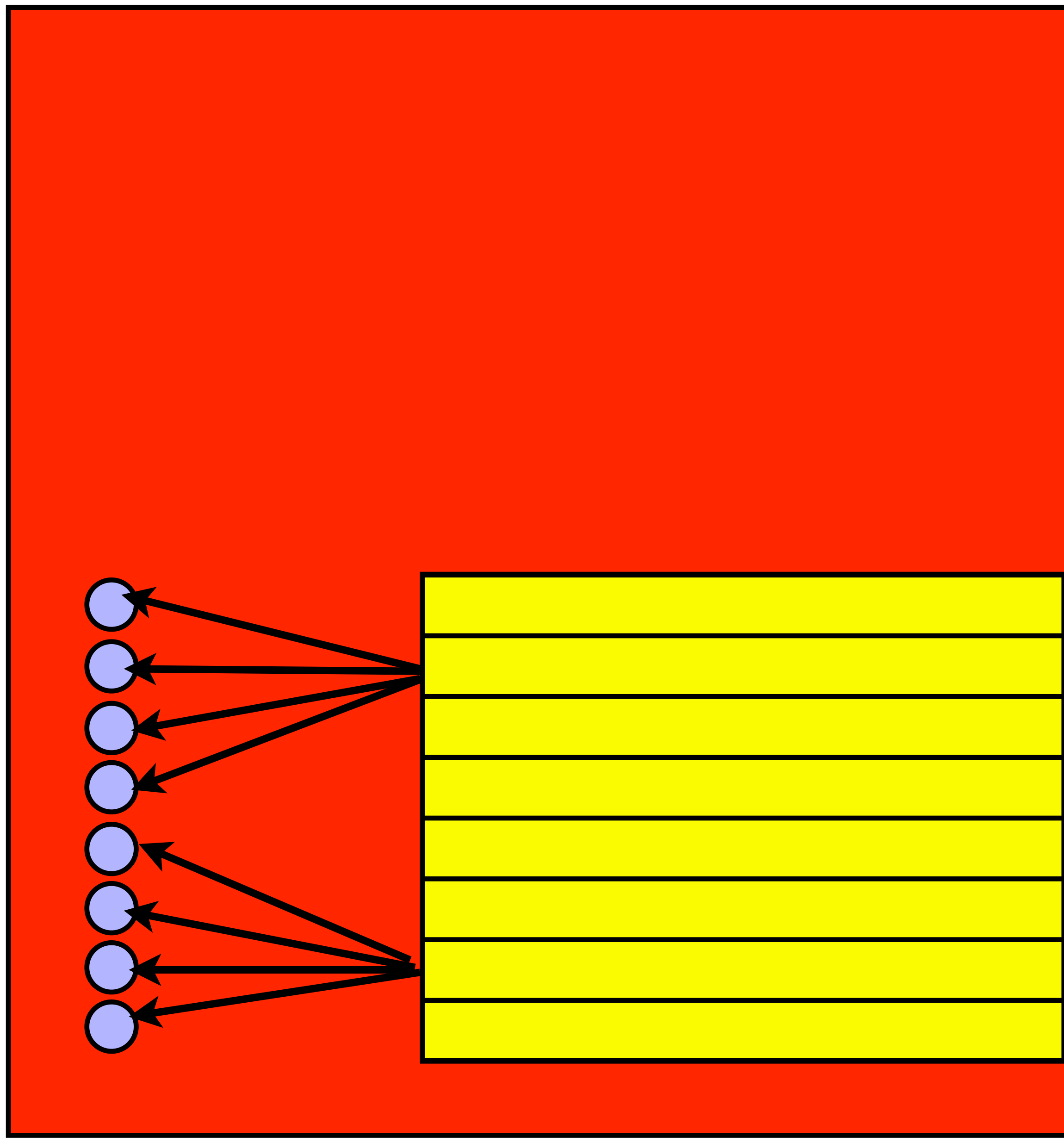
Shared Memory Bank Conflicts

- Each thread in warp accesses different bank: no problem.
- Each thread accesses same one value: ‘broadcast’, no problem.



Shared Memory Bank Conflicts

- Each thread in warp accesses different bank: no problem.
- Each thread accesses same one value: 'broadcast', no problem.
- Multiple threads need data from same bank: conflict. Accesses are serialized.



SM#1

Shared Memory Bank Conflicts

- Imagine 8 banks, and working on an 8xN matrix

Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

CITA|ICAT

SciNet

Shared Memory Bank Conflicts

- Imagine 8 banks, and working on an 8xN matrix
- Row operations are great

Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Shared Memory Bank Conflicts

- Imagine 8 banks, and working on an 8xN matrix
- Row operations are great
- Column operations maximally bad

Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

CITA|ICAT

SciNet

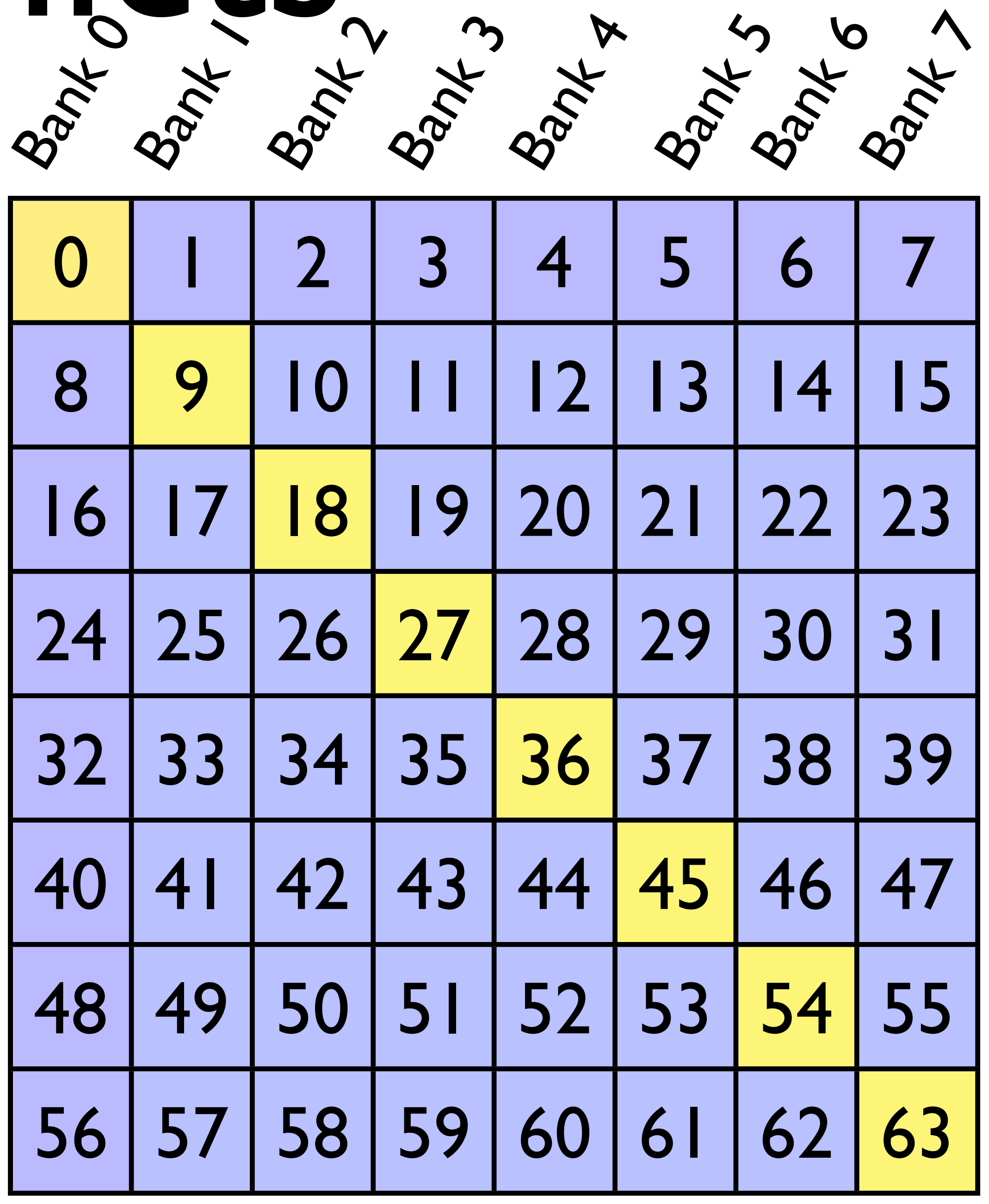
Shared Memory Bank Conflicts

- Imagine 8 banks, and working on an 8xN matrix
- Row operations are great
- Column operations maximally bad
- Solutions
  - Row ops if possible

Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Shared Memory Bank Conflicts

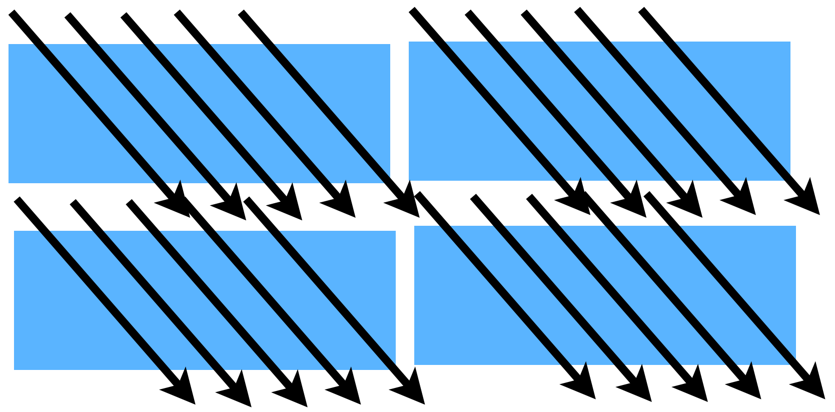
- Imagine 8 banks, and working on an 8xN matrix
- Row operations are great
- Column operations maximally bad
- Solutions
    - Row ops if possible
    - Pad matrix with extra column to stride across banks



Bank 0	Bank 1	Bank 2	Bank 3	Bank 4	Bank 5	Bank 6	Bank 7
0	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47
48	49	50	51	52	53	54	55
56	57	58	59	60	61	62	63

Warps in multi-d blocks

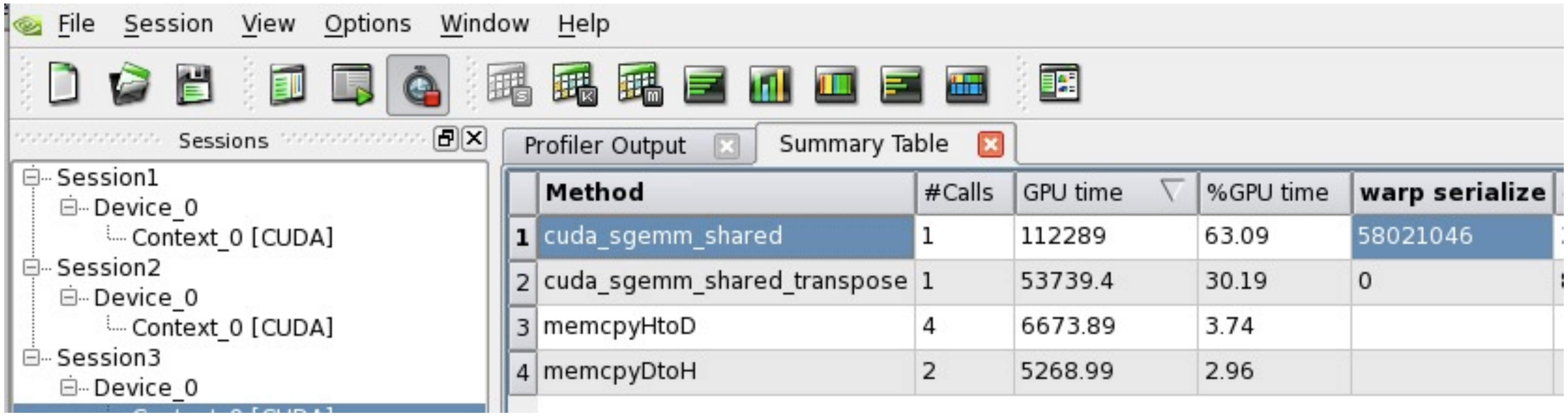
- Easy to see how warps are assigned in 1-d block:
  - First 32 = warp0
  - Next 32 = warp1..
- How done in 2d block?
- C ordering: x first, then y
- blockDim.x = 32:
  - warp 0 : blockDim.y = 0
  - warp 1: blockDim.y = 1..



```
__global__ void cuda_sgemm_shared(const float *ad, const float *bd,
                                  const int n, float *cd) {

    extern __shared__ float shared_data[];

    int loci = threadIdx.x;
    int locj = threadIdx.y;
    int tilesize = blockDim.x;
    int bx = blockIdx.x;
    int by = blockIdx.y;
    int i = threadIdx.x + blockIdx.x*blockDim.x;
    int j = threadIdx.y + blockIdx.y*blockDim.y;
    int k;
    int blockk;

    float *atile = &(shared_data[0]);
    float *btile = &(shared_data[tilesize*tilesize]);

    double sum;
    if (i<n && j<n) {
        sum = 0.;
        for (blockk=0; blockk<gridDim.x; blockk++) {
            /* read in shared data */
            atile[loci*tilesize + locj] = ad[(tilesize*bx+loci)*n + (tilesize*blockk+locj)];
            btile[loci*tilesize + locj] = bd[(tilesize*blockk+loci)*n + (tilesize*by+locj)];
            __syncthreads();
            for (k=0; k<tilesize; k++)
                sum += atile[loci*tilesize + k]*btile[k*tilesize + locj];
            __syncthreads();
        }
        cd[i*n + j] = sum;
    }
    return;
}
```

Striding through matrix w/ slow moving index; Massive bank conflicts if blocksize = warpsize

matmult.cu



	Method	#Calls	GPU time	%GPU time	warp serialize
1	cuda_sgemm_shared	1	112289	63.09	58021046
2	cuda_sgemm_shared_transpose	1	53739.4	30.19	0
3	memcpyHtoD	4	6673.89	3.74	
4	memcpyDtoH	2	5268.99	2.96	

blocksize = 32
= warpsize

```
marten$ ./matmult --matsize=1536 --nblocks=48
Matrix size = 1536, Number of blocks = 48.
CPU  time = 29466.5 millisec, GFLOPS=0.245966
GPU  time = 522.71 millisec, GFLOPS=13.865733, diff = 0.000000.
GPU2 time = 128.905 millisec, GFLOPS=56.225572, diff = 0.000000.
```

4x performance

CITA|ICAT

SciNet

Memory structure informs block sizes:

- By choosing block size in such a way to maximize global, shared memory bandwidth and preloading data into shared, can extract significant performance
- Get your code working first, then use these considerations to get them working fast

```
 ./matmult --matsize=1536 --nblocks=24
Matrix size = 1536, Number of blocks = 24.
CPU  time = 29467.4 millisec, GFLOPS=0.245958
GPU  time = 8.203 millisec, GFLOPS=883.549593, diff = 0.000000.
GPU2 time = 8.122 millisec, GFLOPS=892.361156, diff = 0.000000.
```

- Use tuned code where available (this is still much slower than CUBLAS, MAGMA!)

Homework: Transpose

- Using matmult as a template, write CPU code, then GPU code, which transposes a (float) matrix (square, for simplicity).
- First GPU version: just global memory accesses. (Either read or write necessarily non-coallesced.
- Second version: read tile into shared memory, do both read and write coallesced.
- Time the differences, and use profiler to examine access efficiency. Use (say) 16x16 blocks, and “big enough” that cpu, first gpu version take ~ seconds.
- Note: CPU version also benefits from this “tiling” due to cache