CUDA for Fluid Animation

or

A story of how GPU hardware influences algorithm design decisions

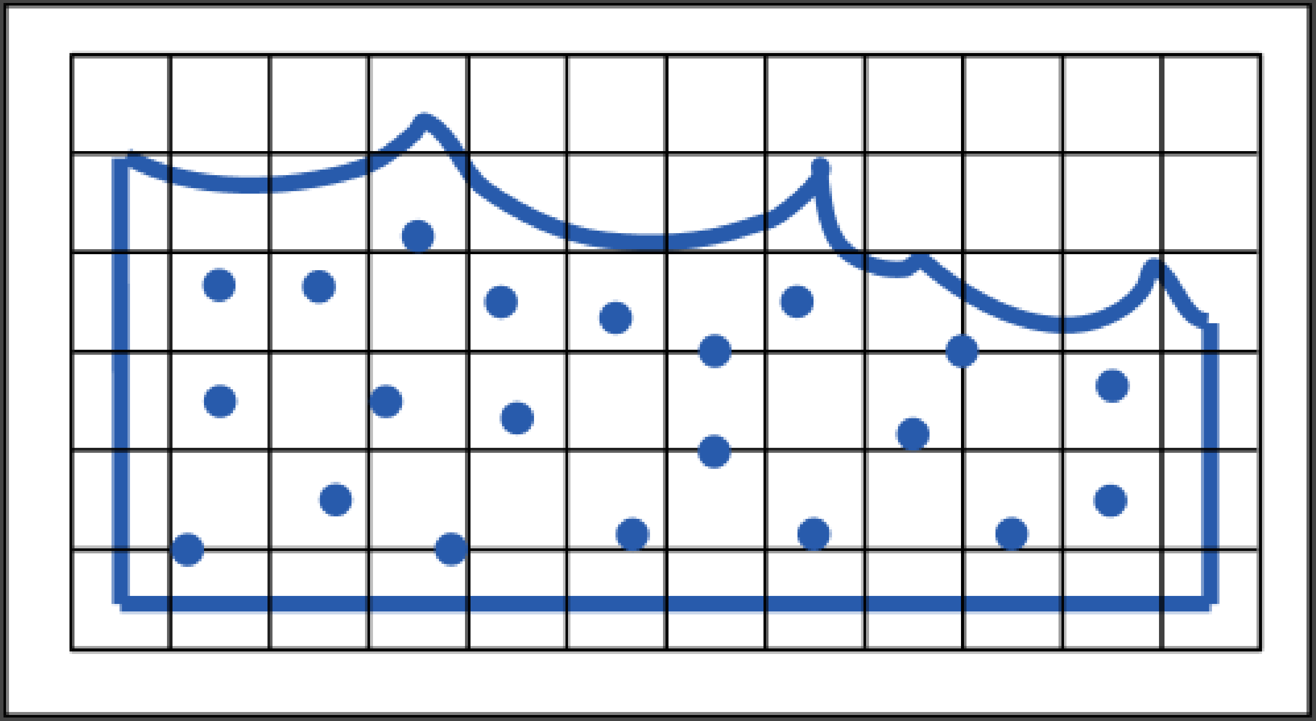
2011-Nov-15

EXOTIC MATTER
NAIAD

Outline

- Introduction to the Problem
- The CPU Implementation
- The GPU Hardware
- GPU Design Influences
  - Parallelism Level
  - Latency
  - Memory: size and latency
  - Memory: bandwidth
- Performance Results
- Future Outlook

Fluid Animation



```
while( true ) {
   move_fluid_particles_around
   transfer_information_to_a_grid
   compute_forces_using_grid
   apply_forces_to_particles
}
```

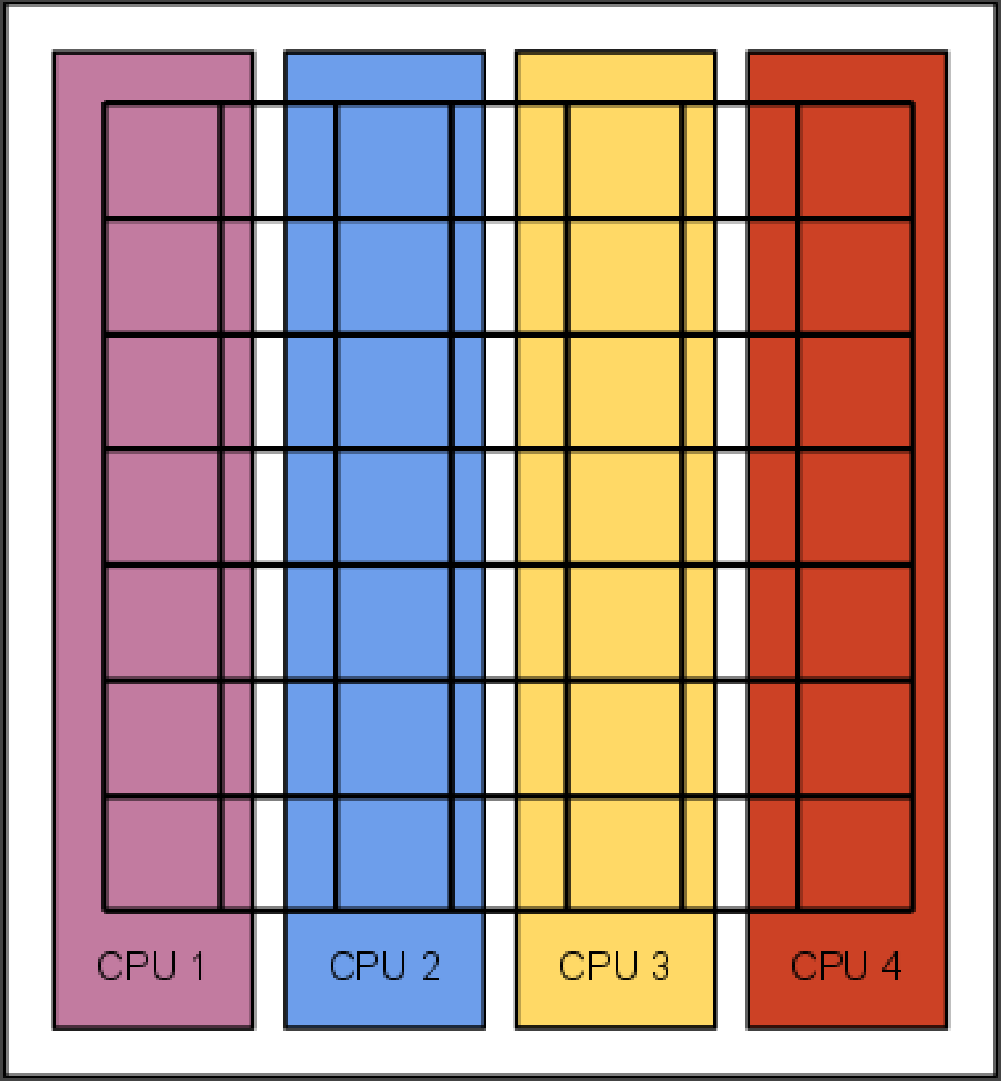
This step is very expensive. Solve a small linear system (Ax=B) at every point in the grid

CPU Implementation

```
parallel_for_each( vertex v )
   out[v] = solve_matrix(A[v],B[v])
```

Results:

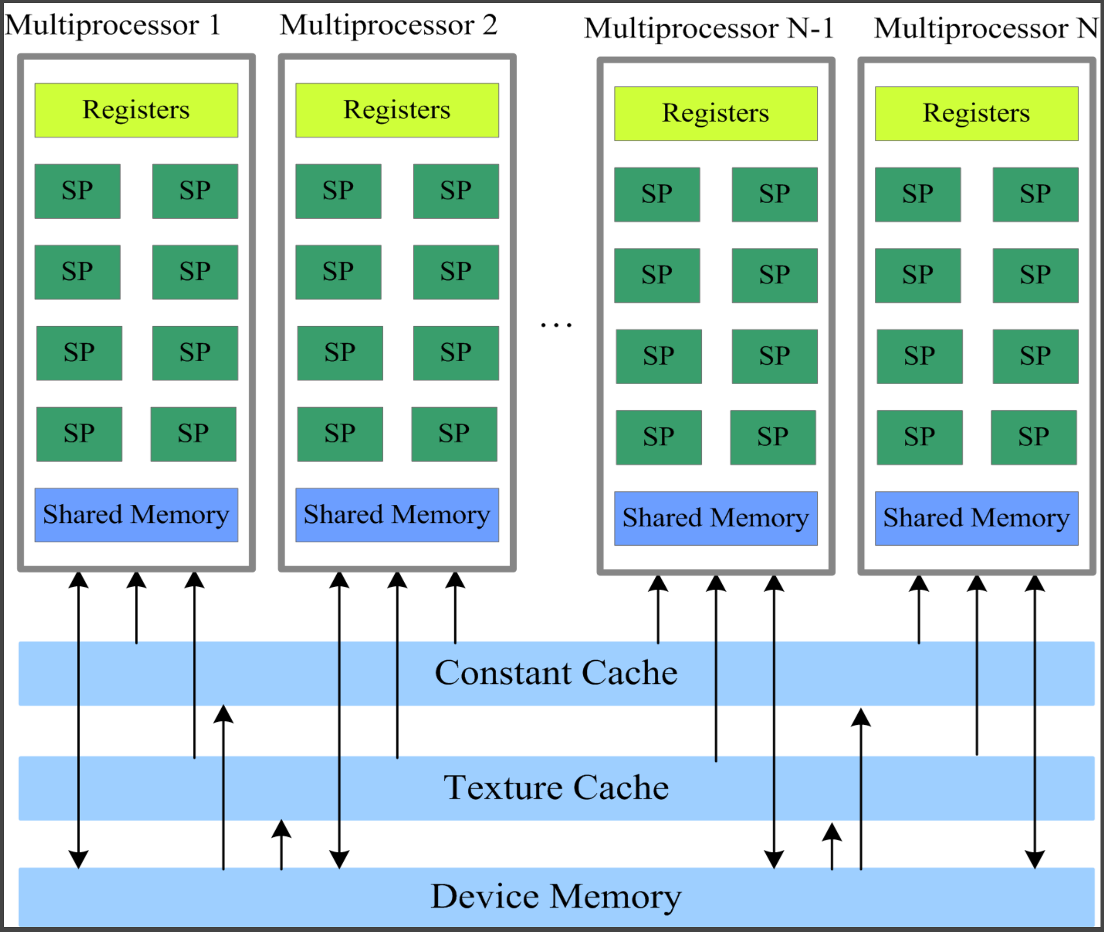
- 512^2 vertices in 4.5 seconds
- 22 seconds if I use AMD optimized solvers on my Intel CPU



Target GPU Hardware

GeForce GTS 450

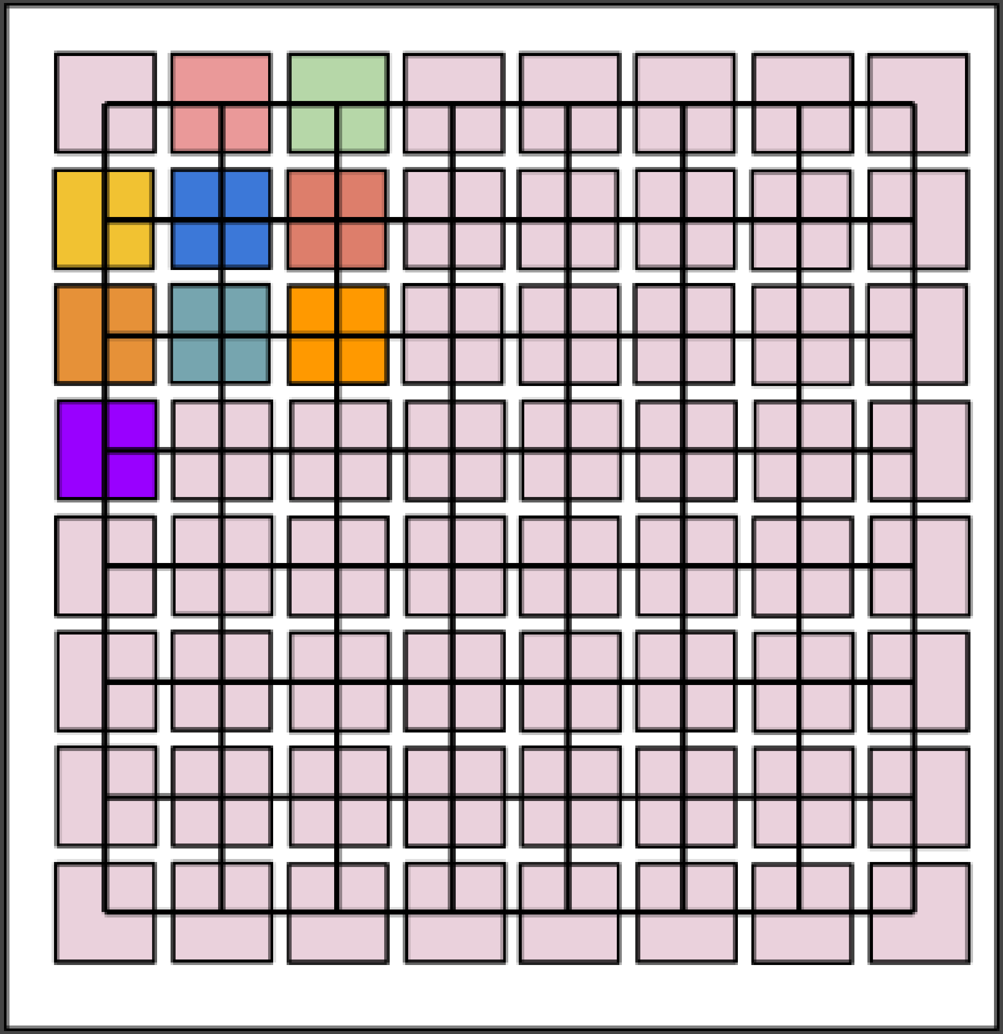
- $130.00 as of Nov. 2011
- 4 multiprocessors (MPs)
- 48 cores per MP
- 192 cores total
- 48 kiB shared memory per MP
- 1GiB device memory



What should each thread do?

Use 1 thread per matrix, because

- this matches the natural level of parallelism for the problem
- using fewer threads is pointless - threads are free
- using more threads is complicated



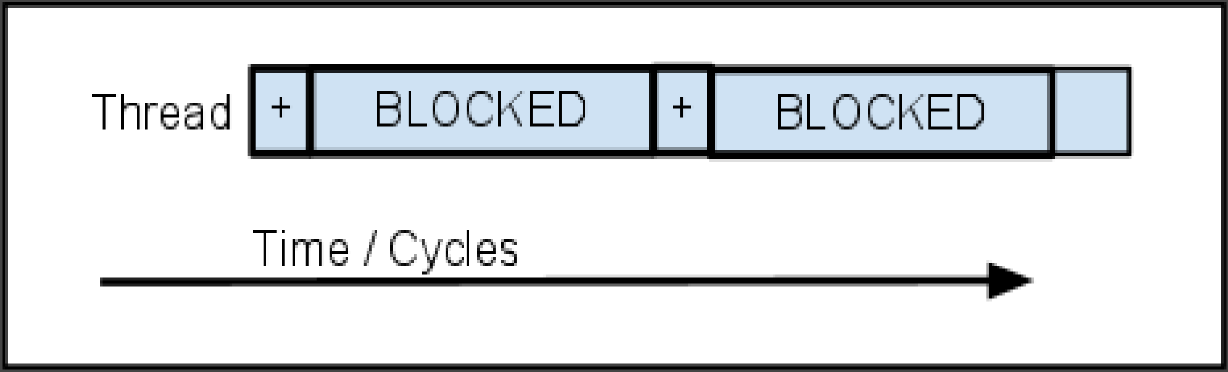
Latency

Best Practices Guide, Section 4.3,
Threads must wait 24 cycles before using an arithmetic result. However, this latency can be completely hidden by the execution of threads in other warps. To hide arithmetic latency completely, multiprocessors should be running at least 192 threads (6 warps).

Latency

Best Practices, Section 4.3,
Threads must wait 24 cycles before using an arithmetic result...

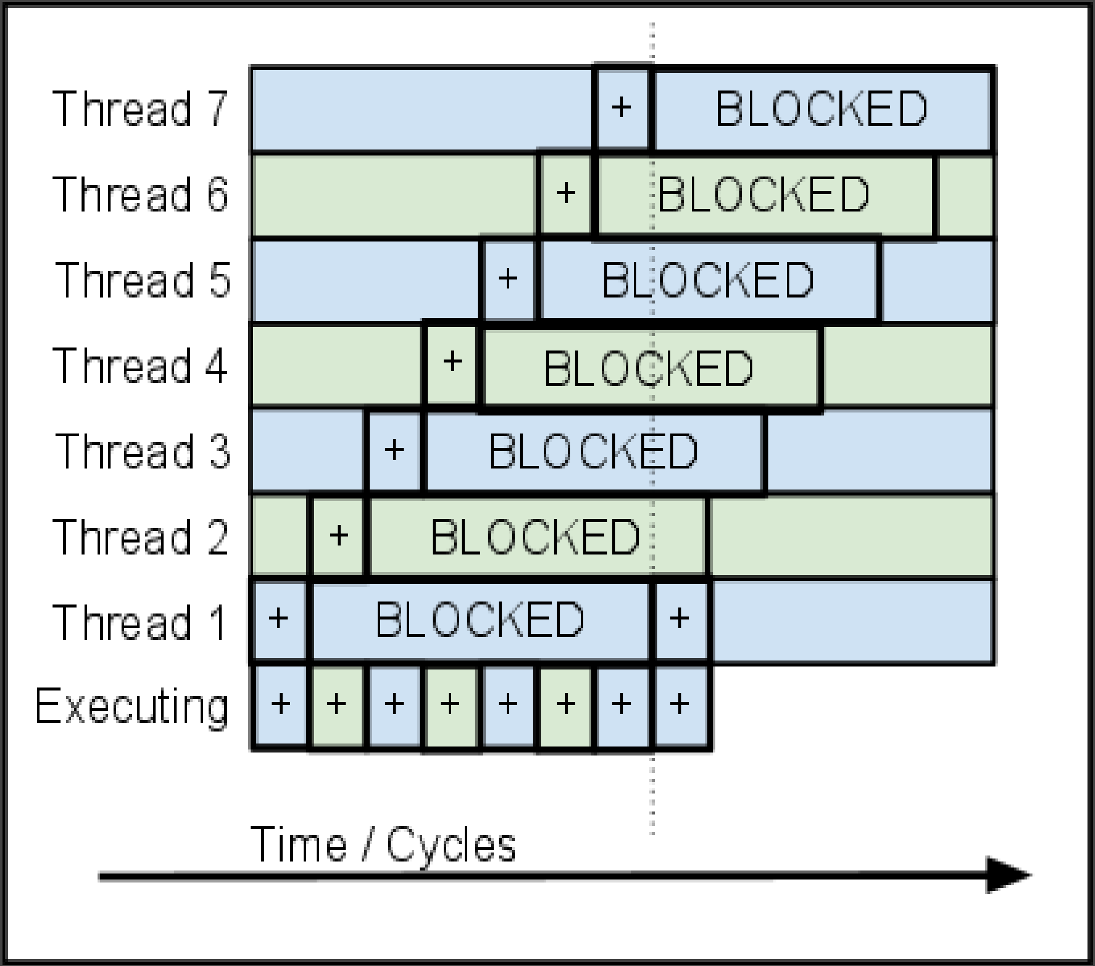
```
float sum(float a, b, c) {
    a = a+b;
    a = a+c;
    return a;
}
```



Hiding Latency

continued....

However, this latency can be completely hidden by the execution of threads in other warps.



Latency Design Conclusion

- We require a lot of threads
- In practice 25 to 50% 'occupancy' is best
- This is 6 to 12 warps per MP on the target hardware
- 12 warps/MP * 32 threads/warp = 384 threads per MP

Is this achievable?

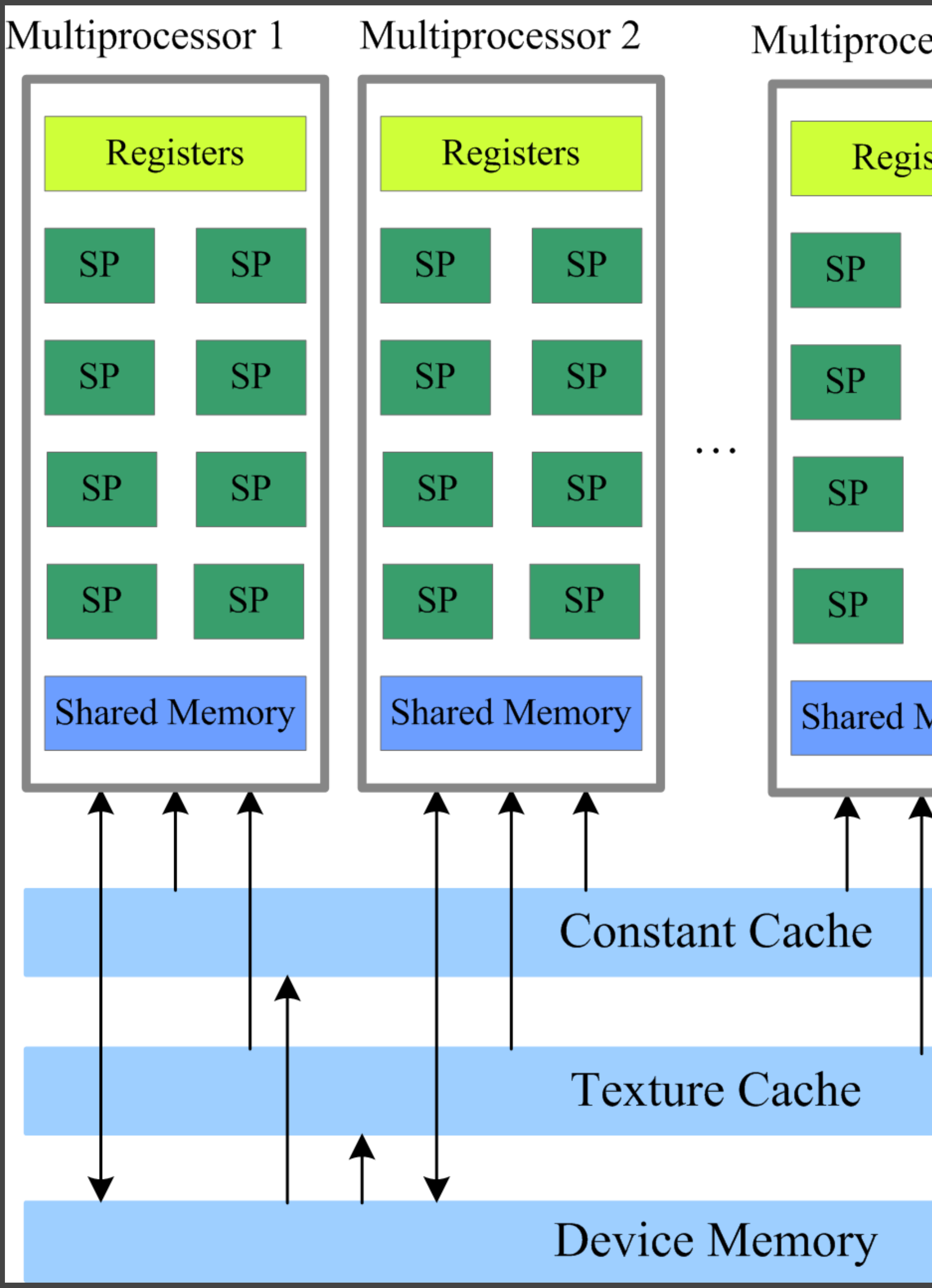
Memory

Shared memory is:

- fast
- small (48 KiB / MP)

Device memory is:

- slow (~500 cycles to read)
- big (1 GiB)

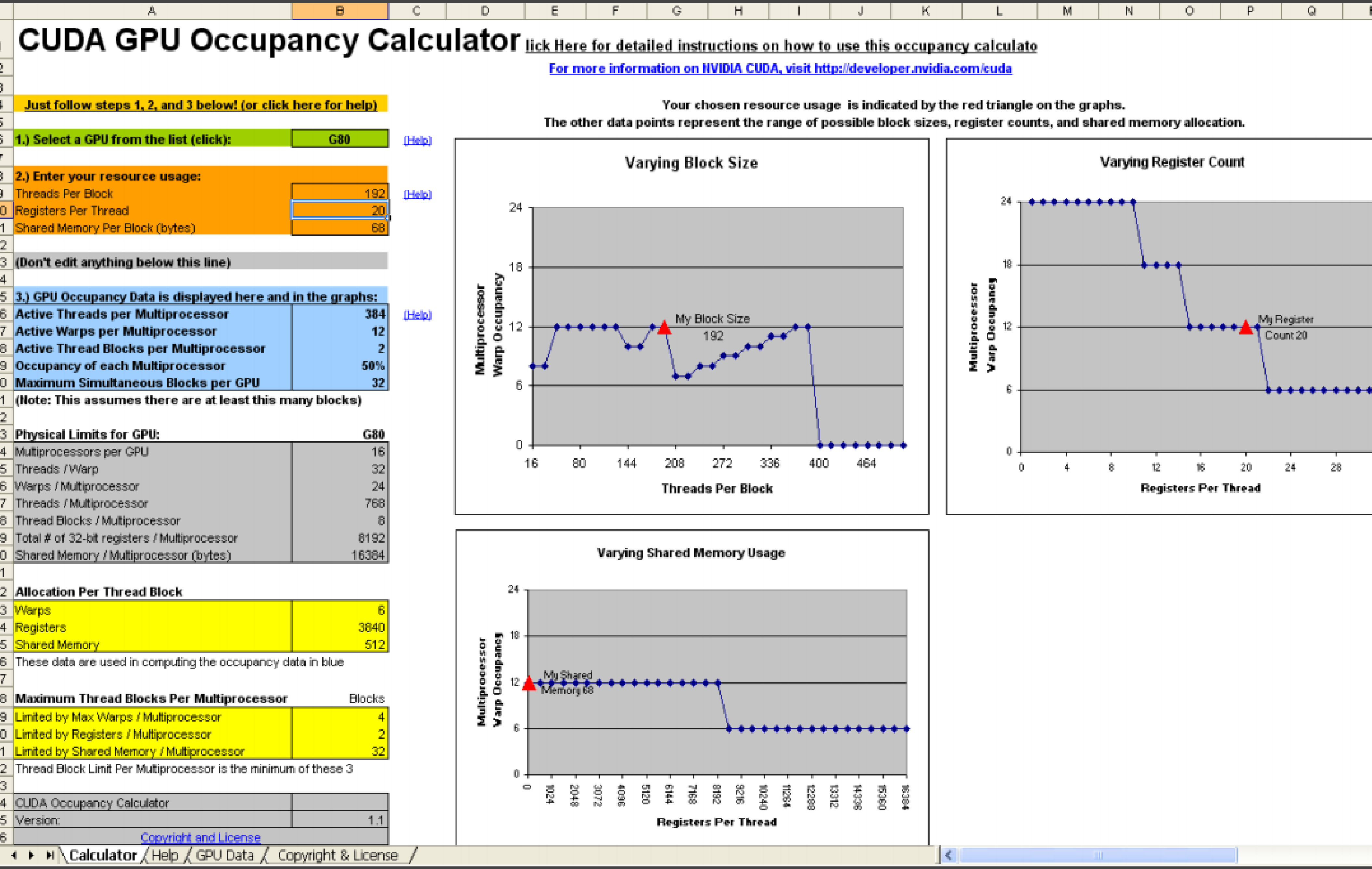


Memory

- Each MP has only 48 kiB of shared memory
- Each matrix is 960 bytes
- So, each MP can hold 50 matrices
- Using shared memory limits us to 50 threads / MP

This conflicts with the latency-driven desire for 384 threads.
What should be done?

Use more threads,
with slow memory,
it's 8 times faster this way.



CUDA GPU Occupancy Calculator lick Here for detailed instructions on how to use this occupancy calculato

For more information on NVIDIA CUDA, visit http://developer.nvidia.com/cuda

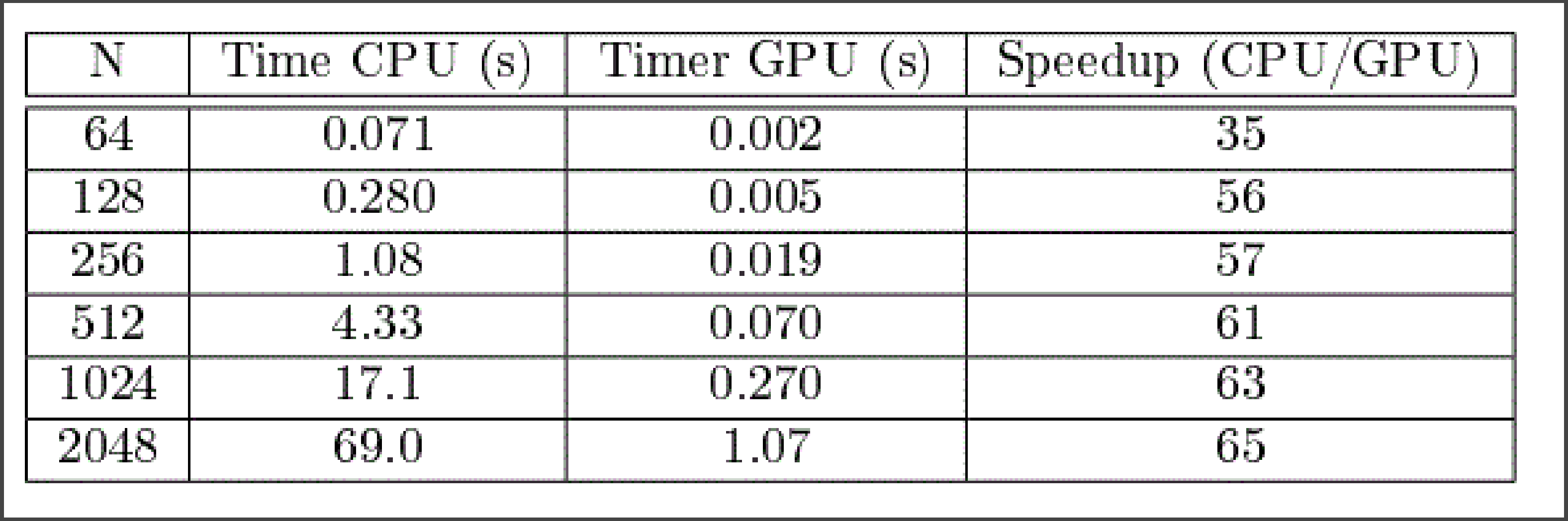
Your chosen resource usage is indicated by the red triangle on the graphs.
The other data points represent the range of possible block sizes, register counts, and shared memory allocation.

Just follow steps 1, 2, and 3 below! (or click here for help)		
1.) Select a GPU from the list (click):	**G80**	(Help)
2.) Enter your resource usage:		
Threads Per Block	192	(Help)
Registers Per Thread	20	
Shared Memory Per Block (bytes)	68	
(Don't edit anything below this line)		
3.) GPU Occupancy Data is displayed here and in the graphs:		
Active Threads per Multiprocessor	**384**	(Help)
Active Warps per Multiprocessor	**12**	
Active Thread Blocks per Multiprocessor	**2**	
Occupancy of each Multiprocessor	**50%**	
Maximum Simultaneous Blocks per GPU	**32**	
(Note: This assumes there are at least this many blocks)		
Physical Limits for GPU:	**G80**	
Multiprocessors per GPU	16	
Threads / Warp	32	
Warps / Multiprocessor	24	
Threads / Multiprocessor	768	
Thread Blocks / Multiprocessor	8	
Total # of 32-bit registers / Multiprocessor	8192	
Shared Memory / Multiprocessor (bytes)	16384	
Allocation Per Thread Block		
Warps	6	
Registers	3840	
Shared Memory	512	
These data are used in computing the occupancy data in blue		
Maximum Thread Blocks Per Multiprocessor	Blocks	
Limited by Max Warps / Multiprocessor	4	
Limited by Registers / Multiprocessor	2	
Limited by Shared Memory / Multiprocessor	32	
Thread Block Limit Per Multiprocessor is the minimum of these 3		
CUDA Occupancy Calculator		
Version:	1.1	
Copyright and License		

Memory Layout and Coalescence

- device memory bandwidth is limited: 58 GiB/s
- each read pulls in a 32, 64, or 128 byte 'page'
- incoherent access pulls lots of pages and consumes the bandwidth
- coherent access is 'coalesced' into just 1 read of a single page
- Getting this right led to a ~15 times speedup over naive code

Performance Results



N	Time CPU (s)	Timer GPU (s)	Speedup (CPU/GPU)
64	0.071	0.002	35
128	0.280	0.005	56
256	1.08	0.019	57
512	4.33	0.070	61
1024	17.1	0.270	63
2048	69.0	1.07	65

The Future

- Other hardware
- Other design decisions
- Other components of the fluid simulation
- Realtime?