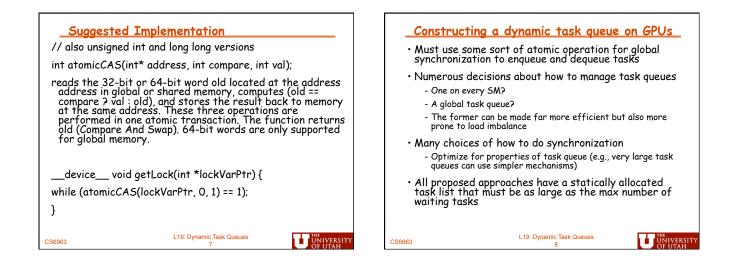
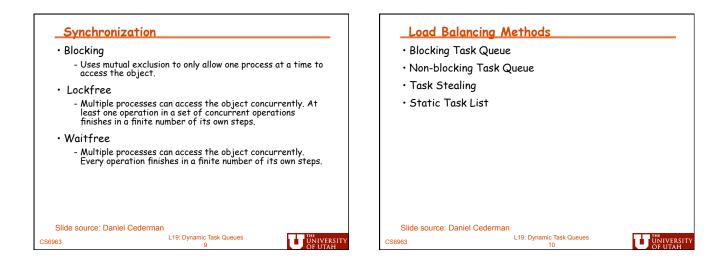
Administrative • Design review feedback - Sent out yesterday - feel free to ask questions L19: Dynamic Task Queues and More Synchronization Deadline Extended to May 4: Symposium on Application Accelerators in High Performance Computing http://www.saahpc.org/ • Final Reports on projects - Poster session April 29 with dry run April 27 - Also, submit written document and software by May 6 - Invite your friends! I'll invite faculty, NVIDIA, graduate students, application owners, ... • Industrial Advisory Board meeting on April 29 - There is a poster session immediately following class - I was asked to select a few projects (4-5) to be presented - The School will spring for printing costs for your poster! - Posters need to be submitted for printing immediately following Monday's class CS6963 UNIVERSITY OF UTAH CS6963

Dry run on April 27 - Easels, tape and poster board provided - Tape a set of Powerpoint slides to a standard 2'x3' poster, or bring your own poster.	 For some of you, with very visual projects, I asked you to think about demos for the poster session This is not a requirement, just something that would enhance the poster session
 Final Report on Projects due May 6 Submit code And written document, roughly 10 pages, based on earlier submission. In addition to original proposal, include Project Plan and How Decomposed (from DR) Description of CUDA implementation Performance Measurement Related Work (from DR) 	 Realistic? I know everyone's laptops are slow and don't have enough memory to solve very large problems Creative Suggestions?

1

Sources for Today's Lecture	Last Time: Simple Lock Using Atomic Updates
 "On Dynamic Load Balancing on Graphics Processors," D. Cederman and P. Tsigas, Graphics Hardware (2008). 	Can you use atomic updates to create a lock variable?
http://www.cs.chalmers.se/~cederman/papers/ GPU_Load_Balancing-GH08.pdf	Consider primitives:
 "A simple, fast and scalable non-blocking concurrent FIFO queue for shared memory multiprocessor systems," P. Tsigas and Y. Zhang, SPAA 2001. 	int lockVar;
(more on lock-free queue)	
 Thread Building Blocks 	atomicAdd(&lockVar, 1);
http://www.threadingbuildingblocks.org/	atomicAdd(&lockVar,-1);
(more on task stealing)	
CS6963 L19: Dynamic Task Queues UNIVERSITY 5 UNIVERSITY	CS6963 L19: Dynamic Task Queues 6 UNIVERSITY

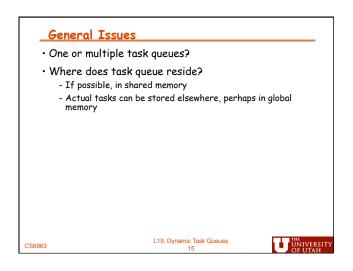




function DEQUEUE(q, id) return q.in[id] ;	Two lists: q_in is read only and
function ENQUEUE(q, task) localtail ← atomicAdd (&q.tail, 1) q.out[localtail] = task	not synchronized q_out is write only and is updated atomically
function NEWTASKCNT(q) q.in, q.out , oldtail , q.tail ← q.out , q.in, q.tail, 0 return oldtail	When NEWTASKCNT is called at the end of major
procedure MAIN(taskinit) q.in, q.out ← newarray(maxsize), newarray(maxsize q.tail ← 0	task scheduling phase, q_in and q_out are wapped
enqueue(q, taskinit) blockent ← newtaskent (q) while blockent = 0 do run blockent = 0 do run blockent blocks in parallel t ← dequeue(q, TBid) subtasks ← doWork(t)	Synchronization required to insert tasks, but at least one gets through (wait free)
for each nt in subtasks do enqueue(q, nt)	
DIOCKS ← NEWTOSKCNT (q) L19: Dynamic Task Queue 963 11	UNIVERS

Blocking Dynamic	Task Queue	
function DEQUEUE(q) while atomicCAS(&q.lc if q.beg != q.end then q.beg ++	ock, 0, 1) == 1 do;	Use lock for both adding and deleting tasks from the queue.
result ← q.data[q.be else result ← NIL q.lock ← 0 return result	:g]	All other threads block waiting for lock. Potentially very inefficient, particularly for fine-grained tasks
function ENQUEUE(q, t while atomicCAS(&q.lc		Ŭ
q.end++ q.data[q.end]← task q.lock←0		
CS6963	L19: Dynamic Task Queues 12	

	function DEQUEUE(q) oldbeg ← q.beg lbeg ← oldbeg while task = q.data[lbeg] == NI L do lbeg ++ if atomicCAS(&q.data[l beg], task, NIL) != task then restart if lbeg mod x == 0 then atomicCAS(&q.beg, oldbeg, lbeg) return task function ENQUEUE(q, task) oldend ← q.end lend ← oldend while q.data[lend] != NIL do lend ++ if atomicCAS(&q.data[lend], NIL, task) != NIL then restart if lend mod x == 0 then atomicCAS(&q.end, oldend, lend)	Idea: At least one thread will succeed to add or remove task from queue Optimization: Only update beginning and end with atomicCAS every x elements.	• Idea: - A set of - When a queues t - Lots and	rovided in paper independent task queues. task queue becomes empty, it goes out of find available work t lots of engineering needed to get this rk on this is in Intel Thread Building Bl	s right
--	--	---	---	--	---------



4