Competition => Collaboration

a runtime-centric view of parallel computation

Simon Kahan
skahan@cs.washington.edu
Competition

Multiple entities in contention for limited, indivisible resources or opportunities

©2009, 2013 Simon Kahan
Direct Mitigation Techniques

- Take turns
- Share
- Find more dolls
Direct Mitigation Techniques

- Take turns
  - Mutual Exclusion

- Share
  - Transactions

- Find more dolls
  - Replication (eg, of Data Structures)
Direct Mitigation Techniques

- Take turns
  - Mutual exclusion
  - Delay is linear in concurrency: does not scale
- Share
  - Transactions
  - Aborted work is up to quadratic in concurrency: does not scale
- Find more dolls
  - Replication (eg, of Data Structures)
  - Cost ~ maximum concurrency sustained + coherency overheads: does not scale
Collaboration

Entities align to reduce contention, increase throughput.

©2009, 2013 Simon Kahan
Transform competition to collaboration?

Why won’t these people collaborate?!
Are computers better collaborators?

©2009, 2013 Simon Kahan
MTA-2 Processor

Every clock cycle, a ready instruction may begin execution...

Stream1
Stream2
Stream128

(M A C)
load, store,
int_fetch_add

+ - * /, etc.

Arithmetic
Pipeline
Simplified Cray/Tera MTA-2 System Architecture

4,000 Active Threads

CPU

CPU

CPU

\[\cdots\]

Memory

Be Parallel or Die.
Memory Allocation

OS

sbreak()

Heap

malloc(), free()

Application
Parallel Memory Allocation

OS

sbreak()

Heap

malloc(), free()

Application

malloc(), free()
Replication for Concurrency
Increase heap size to lower sbreak rate

Q: What’s wrong with this picture? A: O(P²) wasted space!
Can collaboration help?

• Idea: apply the ticket line trick!
  – tasks need to “find” each other
  – aggregate their requests into one
  – one “master” task continues; other waits
  – until master finds heap uncontended, repeat process
  – master locks heap, fulfills request, unlocks heap
  – master recursively splits and awakens waiters

• Simon Kahan and Petr Konecny. 2006. "MAMA!": a memory allocator for multithreaded architectures. PPoPP '06.
Combining Funnels

Concurrent Asynchronous Individual Malloc and Free Requests

Concurrency: F

Time: \( \lg F \)

“Funnel”: combining data structure

Aggregate Requests of Size at most F served serially.
(Output rate is at most a constant.)

©2009, 2013 Simon Kahan
Aggregates: Pennants for speed

Single requests (Pennants of order 0)

• Merge is 2 ops:
  - T2.left = T1.right
  - T1.right = T2

• Balanced
  - Unlike linked lists, supports parallel traversal

• Unique representation
Tree-Heap

while (int_fetch_add(&sem, 1)) try_combine();
heap_op(); sem = 0;

Allocate tries for corresponding slot; if empty, marches to right.
Free tries for corresponding slot; if full, combines and carries.
It’s just binary arithmetic! Worst-case O(log N); Average O(1)

©2009, 2013 Simon Kahan
Instructions vs Delay

MTA Malloc 10 cpu highest (instructions)

MAMA! 1 cpu is highest
Original MTA malloc vs MAMA

220 MHz MTA-40, 100 streams per processor

Figure 11. Microseconds per malloc

©2009, 2013 Simon Kahan
General Combining Scheme

Asynchronous (Competitive)
• Arbitrary # computations
• Any number of threads
• Timing of interaction arbitrary
• Chaos!

Synchronous (Collaborative)
• Single computation
• Number of threads is explicit
• Synchronized, exclusive access to data
• Order!

Data Structure
Conclusion

• Concurrency often creates competition.
• Competition indicates duplication in need.
• Serializing, transacting, replicating -- may only mitigate competition
• Consider transforming competition to collaboration, aligning common need to get there faster.