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

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.

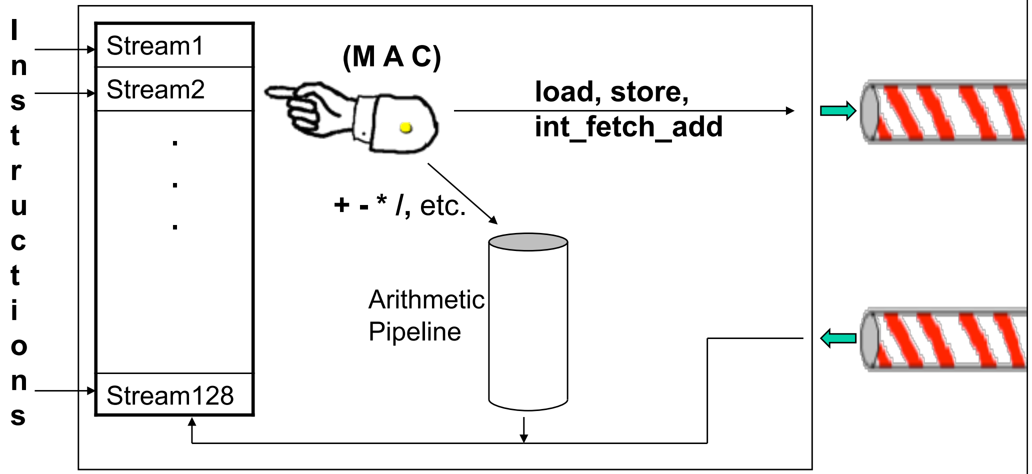
Transform competition to collaboration?

Why won't these people collaborate ?!

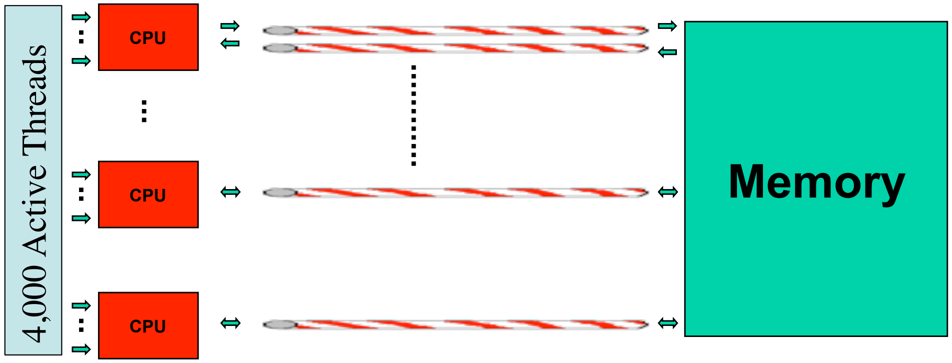
Are computers better collaborators?

MTA-2 Processor

Every clock cycle, a ready instruction may begin execution…

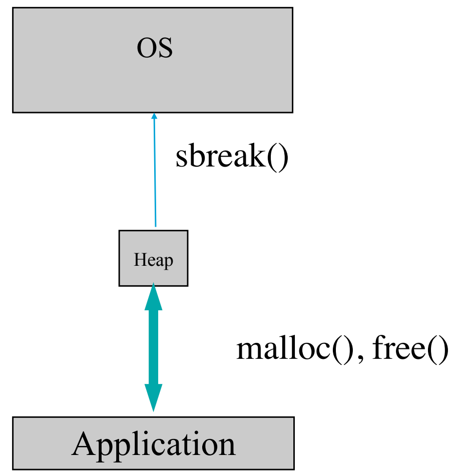


Simplified Cray/Tera MTA-2 System Architecture

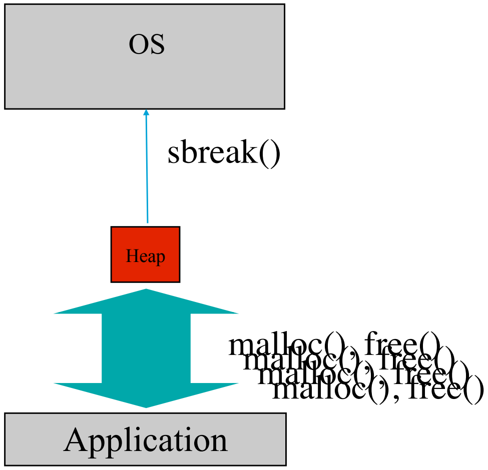


Be Parallel or Die.

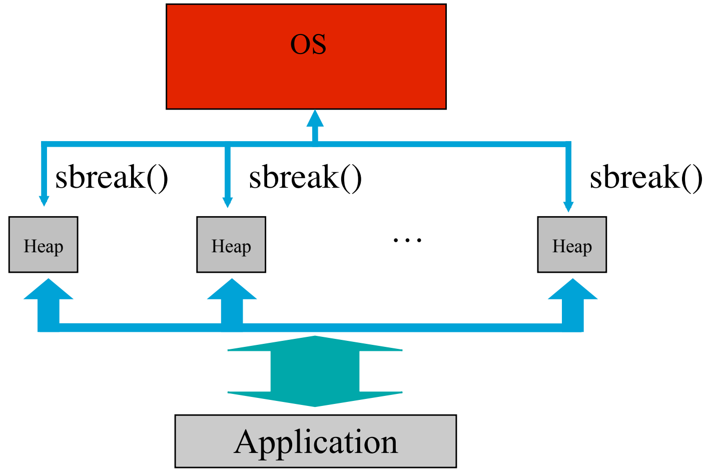
Memory Allocation



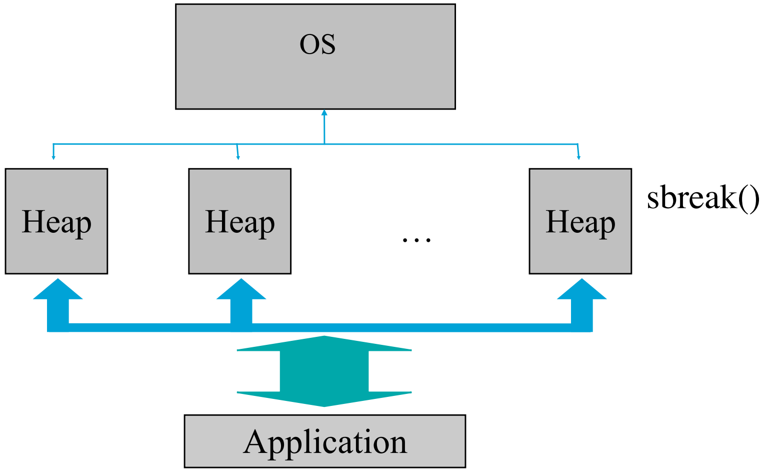
Parallel Memory Allocation



Replication for Concurrency



Increase heap size to lower sbreak rate



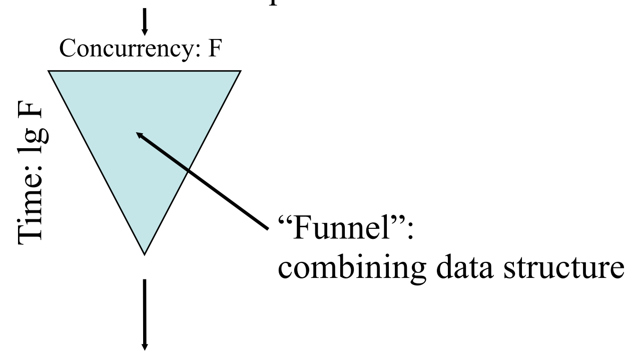
Q: What's wrong with this picture? A: $O(P^2)$ wasted space!

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

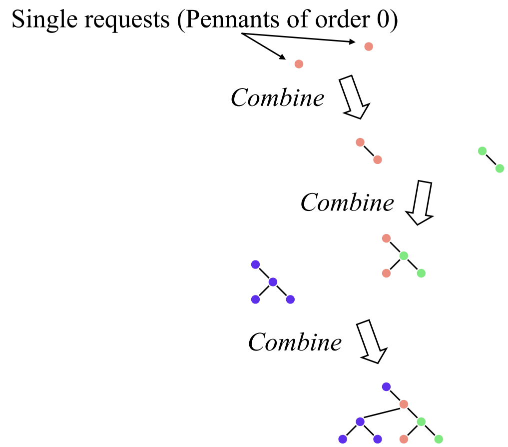
Concurrent Asynchronous Individual
Malloc and Free Requests



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

See: “Combining Funnels: a Dynamic Approach to Software Combining”, Nir Shavit, Asaph Zemach, 1999

Aggregates: Pennants for speed



- Merge is 2 ops:

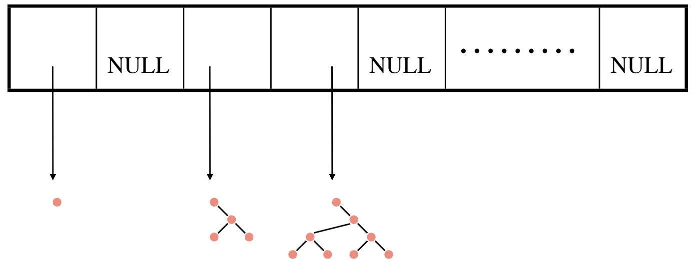
  T2.left = T1.right

  T1.right = T2
- Balanced

  Unlike linked lists,
  supports parallel traversal
- Unique representation

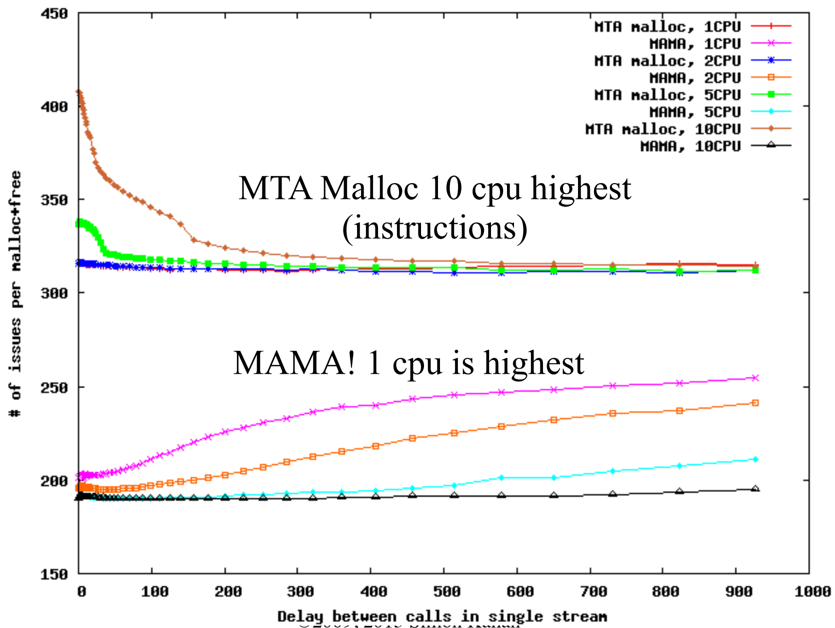
Tree-Heap

```
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)

Instructions vs Delay



Original MTA malloc vs MAMA

220 MHz MTA-40, 100 streams per processor

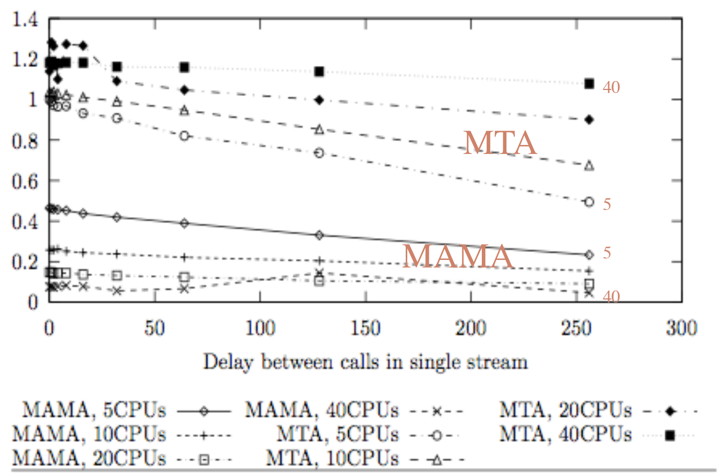


Figure 11. Microseconds per malloc

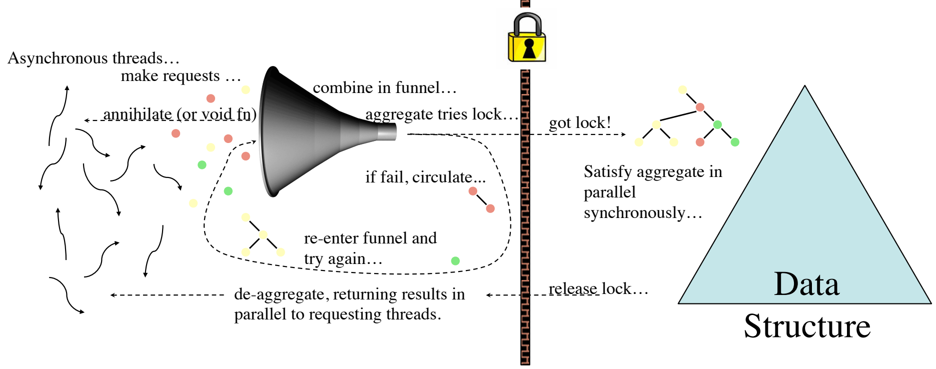
General Combining Scheme

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!



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.