Flat Combining and the Synchronization-Parallelism Tradeoff

Danny Hendler, Ben-Gurion University
Itai Incze, Tel-Aviv University
Nir Shavit, Tel-Aviv University
Moran Tzafrir, Tel-Aviv University

SPAA '10 Proceedings of the 22nd ACM Symposium on Parallelism in Algorithms and Architectures

Hannes Payer, University of Salzburg, January 2011

Research Problem

- Performance of concurrent data structures
- Traditionally: provide parallelism via fine grained synchronization
  - It has been shown in several studies that finely synchronized data structures outperform data structures protected by a single global lock
- Is the above assumption true in general?

Answer

- No, because of synchronization overhead!

Flat Combining - General Idea

- Ingredients:
  - Original data structure
  - Global lock
  - Publication list and mapping of threads to thread-local publication records
- A thread performs a data structure operation in the following way
  1) Acquire a global lock
  2) Learn about all concurrent access requests
  3) Perform the combined requests of all pending requests

Flat Combining - Details

Algorithm 1: Flat Combining Generic Structure

```
shared object 𝒪, int lock init 0
shared array combine[1..maxT] of int init ⊥
Input: <opcode,params>
combine[PID]=<opcode,params>
while true do
    if (lock=1) ∨ (test-and-set(lock)=1) then
        yield()
        if (resp=combine[PID])≠<opcode,params>
        then return resp
    else
        ScanCombineApply(combine, 𝒪, ϕ)
        int response=combine[PID]
        lock=0
        return response
```

Flat Combining - Details

1) Write data structure operation and paraments (if any) to be applied to the shared data structure in the thread-local publication record

2) Check if global lock is taken

   - If so, spin on the publication record waiting for a response to the invocation.
   - Once in a while check if the lock is still taken
   - If response is available in the publication record: reset the thread-local publication record to null and return response

Flat Combining - Details

3) If global lock is not taken, attempt to acquire it and become a combiner.

- Otherwise return to 2)

4) Execute scanCombineApply()

- Is specific for different data structures
- Scan over publication list, comine requests, and return results of the invocations
- Guaranteed to be Wait-free
- Release the global lock

Flat Combining - Details

- Publication list can grow and shrink dynamically
  - Multiple ways to do this but they require synchronization operations
- A static publication list size offers the best performance

Flat Combining Queue and Stack

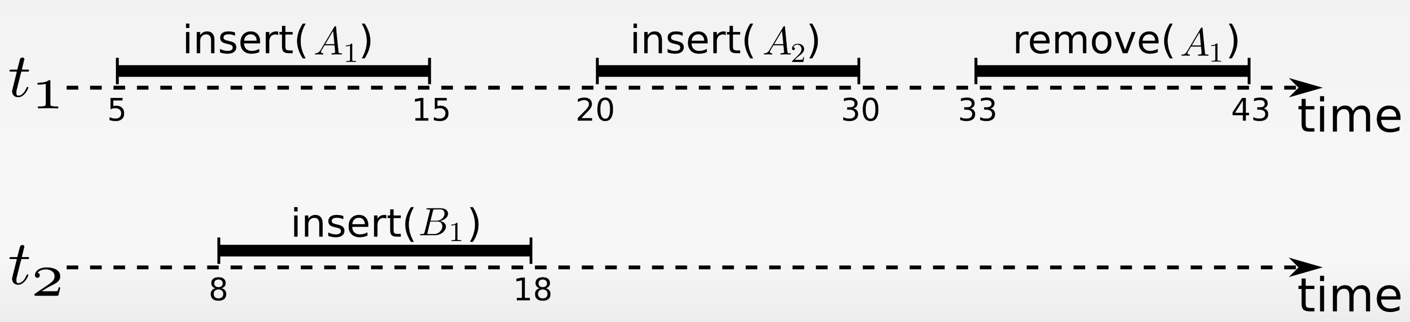
- ScanCombineApply()
  - Queues and stacks have an inherent sequential bottleneck that is difficult to overcome
  - A temporary list is used to combine pending requeusts
  - A non-empty temporary list is in the end concatenated with the original data structure

Flat Combining Skiplist

- ScanCombineApply()
  - Each node consists of a key and a list
  - RemoveSmallestK()
    workload is comparable to a single removel
  - CombinedAdd()
    sort pending requeusts and perform just a single pass through the list

Flat Combining Correctness

- Linearizability
  - Correctness condition of shared data structures
  - Each operation takes effect instantaneously at some (linearization) point between its invocation and response
  - Proof: show that a linearization can be found for each execution on the data structure



Flat Combining Correctness

- Proof outline:
  - The global lock serializes all data structure operations
  - Since there is just a single combiner thread at each point in time, operations are ordered sequentially
  - Threads are blocked unless their data structure operation is applied

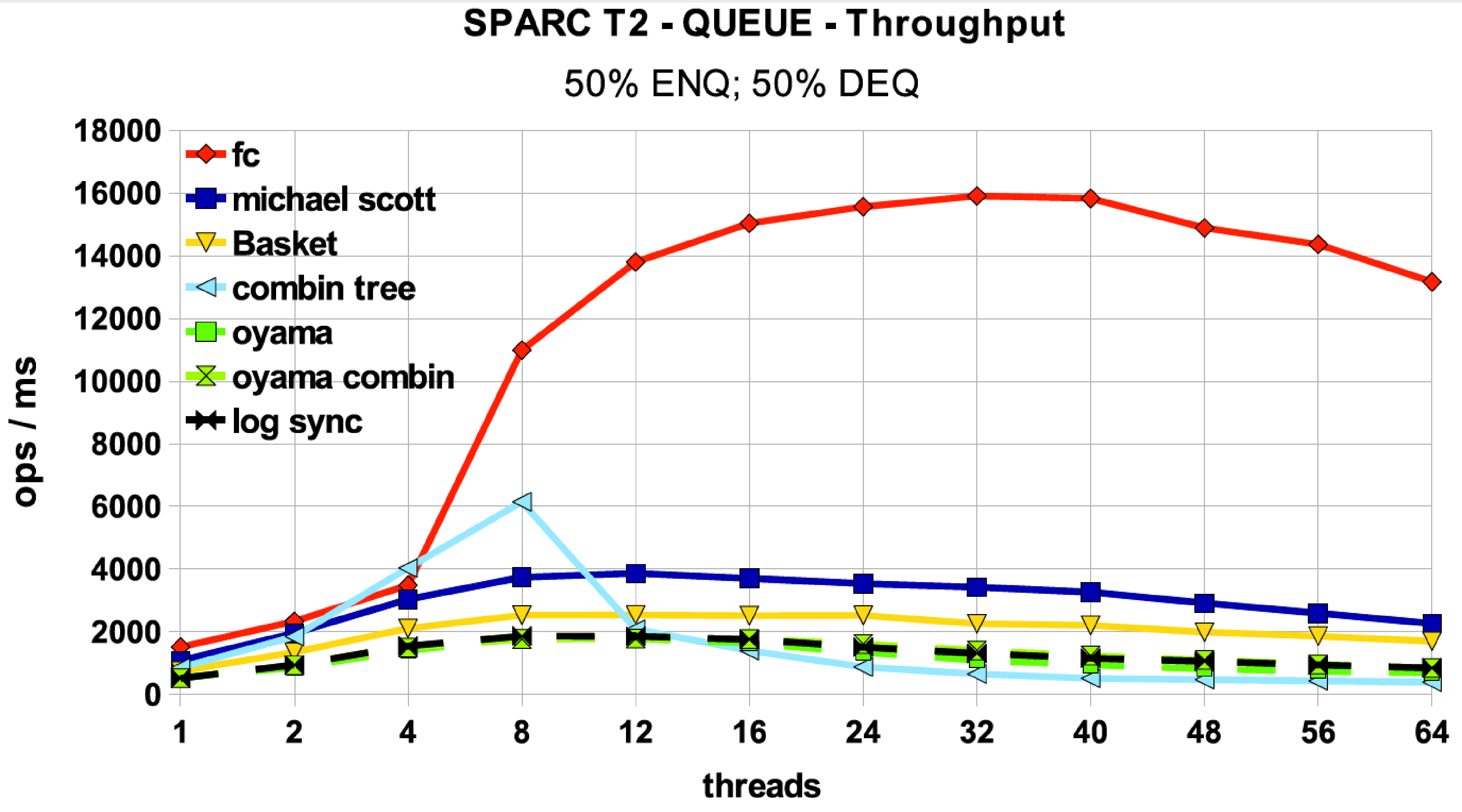
Flat Combining Progress

- Flat combining is starvation free
  - ScanCombineApply is wait-free
- Proof outline:
  - The data structure operation of a single thread is performed by a current combiner thread or by a subsequent combiner thread

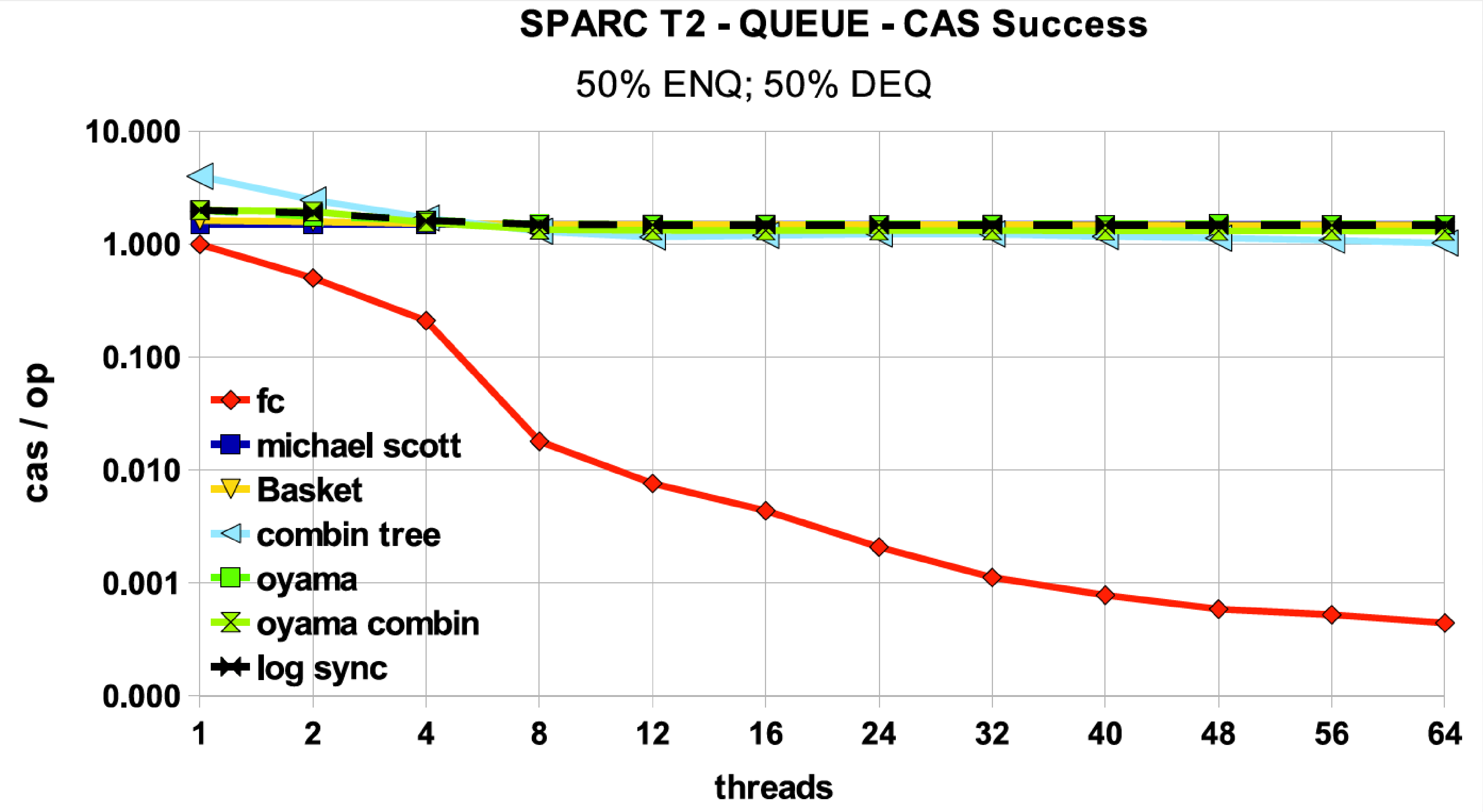
Experimental Setup

- 128-way Enterprise T5140 server machine running Solaris
- 2-chip Niagara system, each chip has 8 cores that multiplex 8 hardware threads each and share an L2 cache
- Hoard memory allocator to reduce system jitter

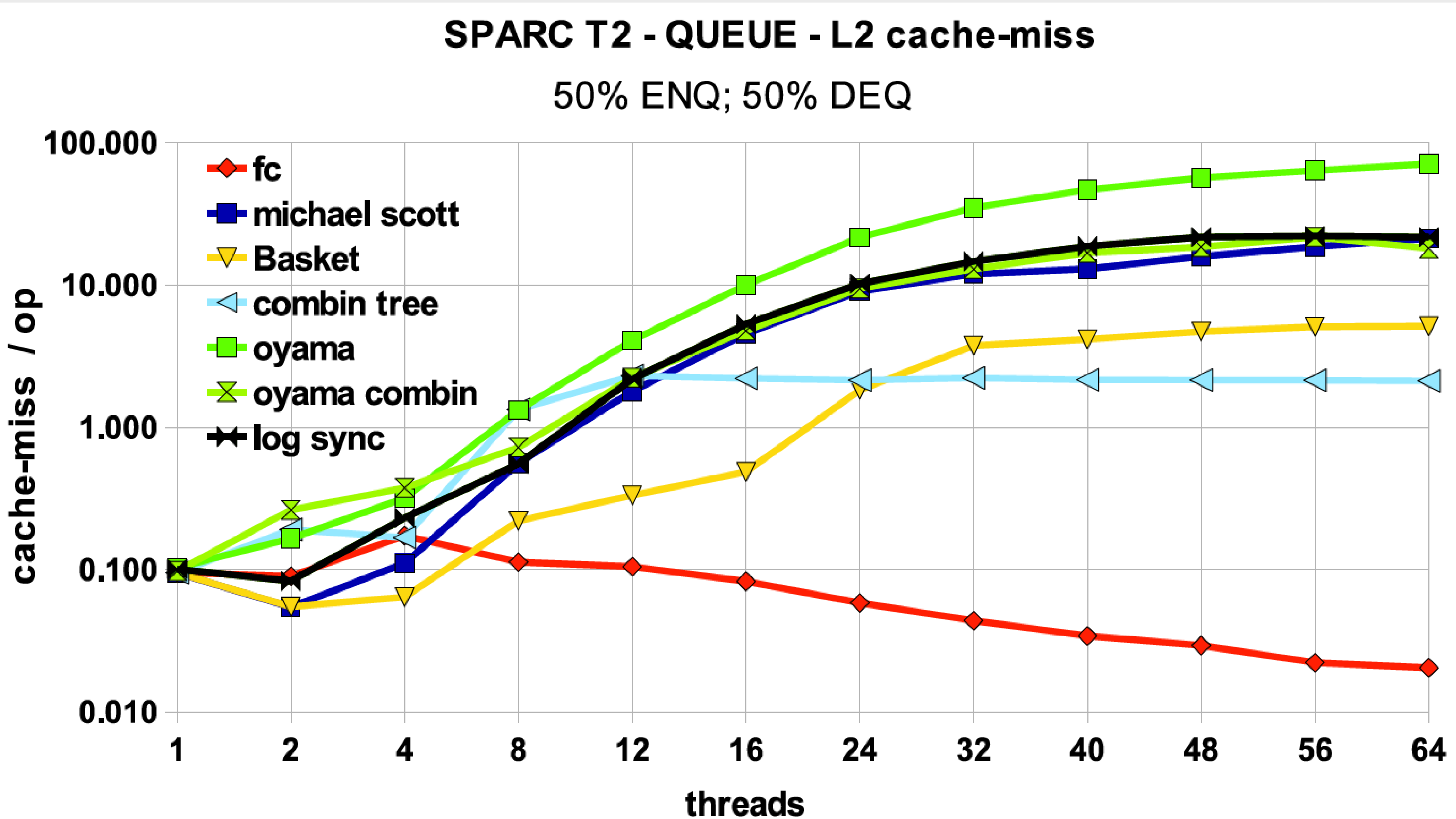
Performance Evaluation



Performance Evaluation



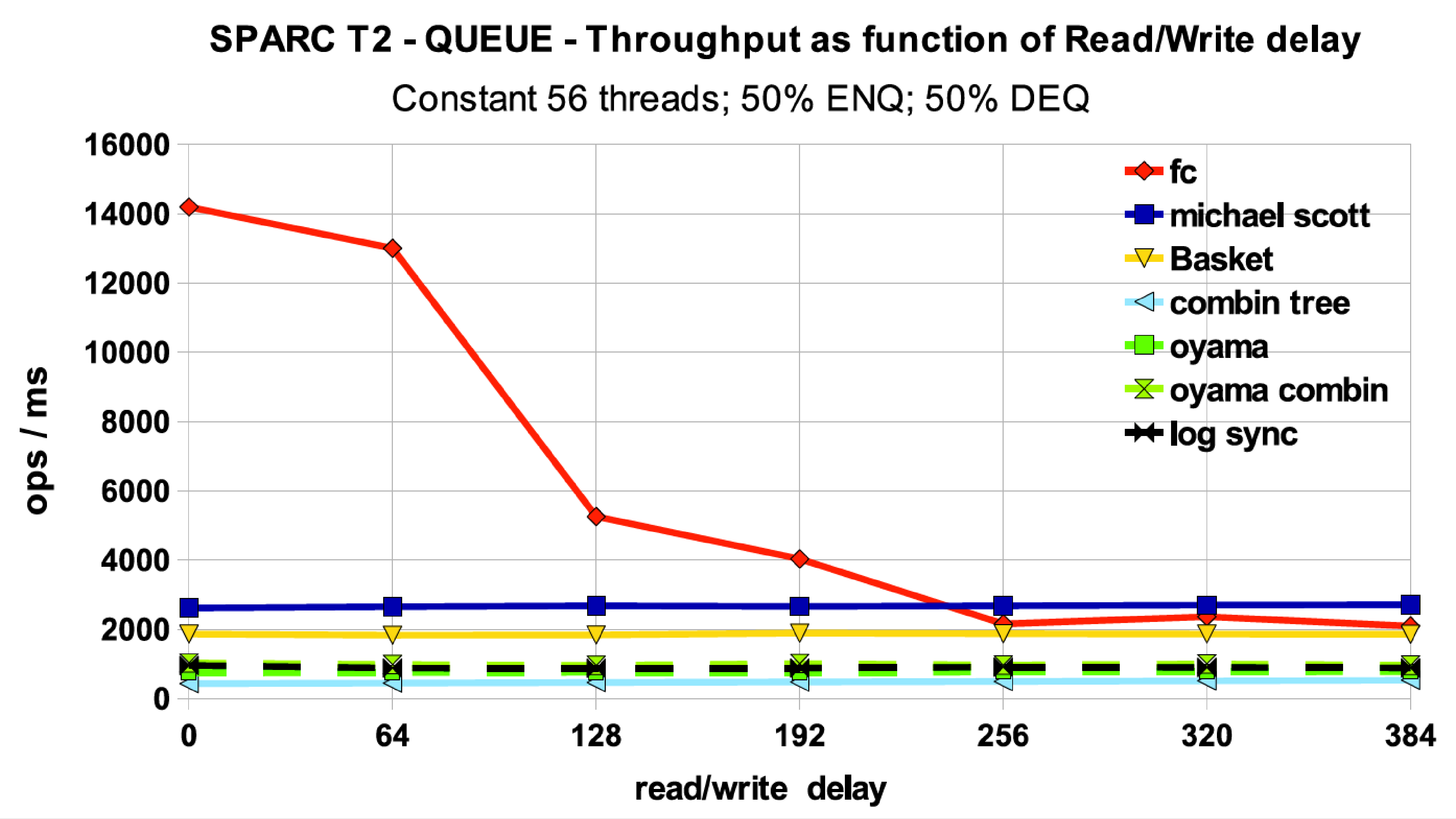
Performance Evaluation



Why does Flat Combining work?

- Reduces synchronization overhead on shared data structure
- Reduces the overall cache invalidation traffic on the data structure
- Locality
- Items are cached
- Take advantage of re-ordering of operations and applying them at the same point in time

Performance Evaluation



Questions?