Funnel Sort*: Cache Efficiency and Parallelism

Paul Youn, 6.895 Final Project
(5689FPPYaaceijllnnoortuu)

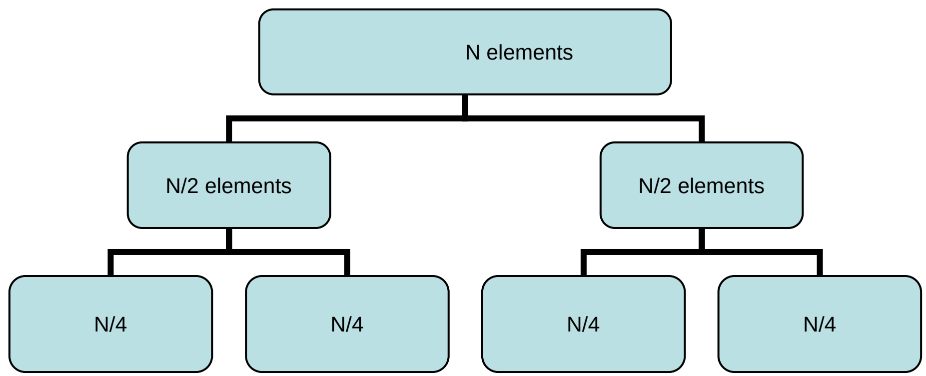
* Developed by Frigo, Leiserson, et. al.

Outline

- Cache Efficient vs Cache Oblivious
- Funnel Sort: How it achieves cache efficiency
- Performance:
  - Serial Execution
  - Cache Performance
  - Parallel Implementation
- Future Possible Work
- Conclusions

Cache Efficiency and Sorting

- Cache Efficiency: Many sorting algorithms are cache oblivious, but not cache efficient. For example, a 2-way recursive sorting algorithm (such as Quick Sort or Merge Sort).



- At every level all *N* elements are processed, which means all *N* elements are loaded into cache. If loaded one line at a time, for every *L* (size of cache line) elements loaded into cache, there is 1 cache miss: 1/*L* amortized cache miss per element.

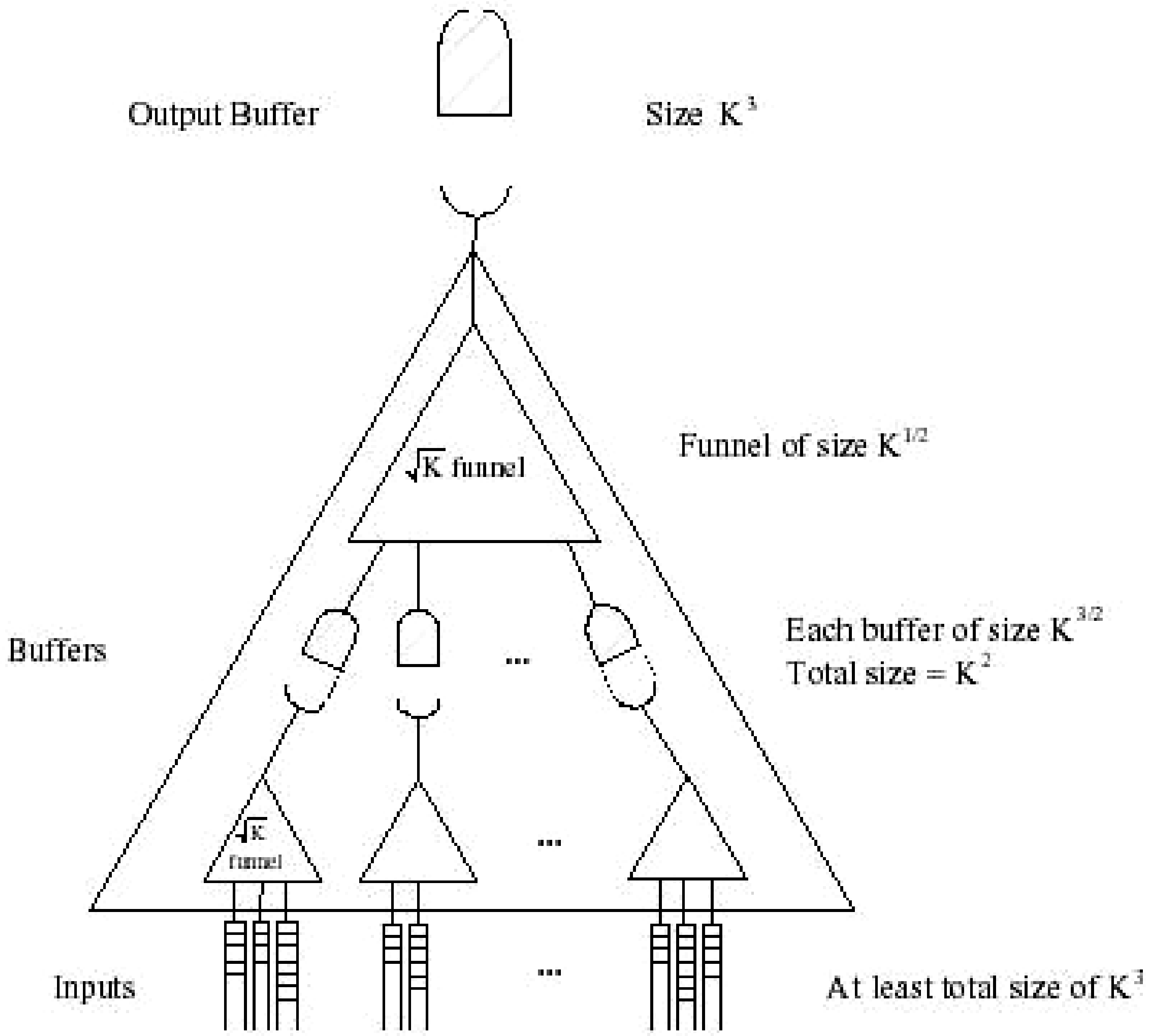
Cache Performance

- Implies an $O(n/L \lg n)$ cache miss bound.
- If Z is the number of lines in a cache, Professor Bender's lecture showed cache aware $O(n/L * (\log n)/(\log Z))$ cache miss sorting algorithm by doing an $O(Z)$-way merge sort.
- Works by decreasing the height of the tree.
- Can we achieve this bound with a cache-oblivious algorithm? Yes!

Funnel Sort

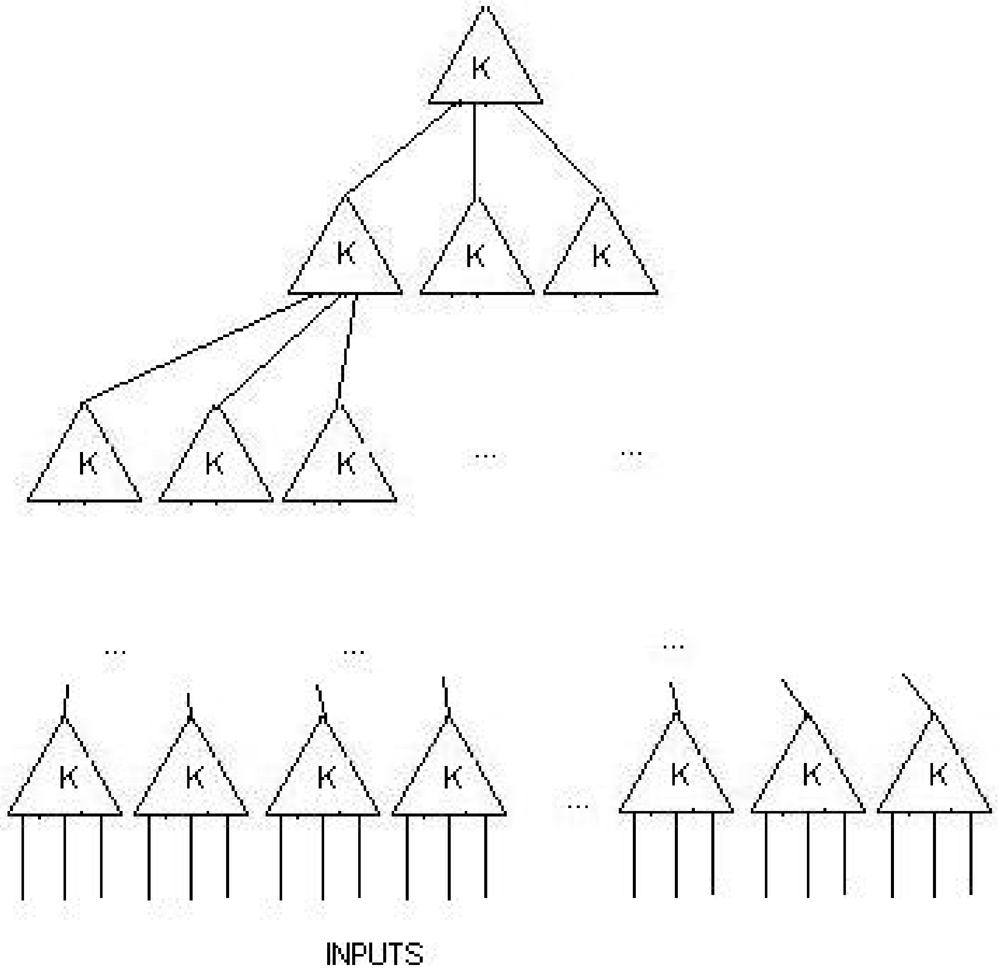
- Try to get as close to a Z-way merge as possible.
- Intuition: Want to recursively lay out a K-way merge (lets call it a K-funnel) to consist of smaller funnels.
- Important that all K-funnels be the same size no matter what location in the sort tree!

- Note that all *K*-funnels will be of the same size regardless of the size of inputs.
- At some point, *K* is small enough that the *K*-way merge fits into cache.



When *K* fits in cache

- For some *K*, the entire *K*-funnel will fit in cache.
- Now, think of the original problem as a series of *K* merges, where *K* is close to *Z*.



Funnel Sort cache efficiency

- For the appropriate constants for the size of buffers, achieve $O(n/L * (\log n)/(\log Z))$ cache misses for this cache-oblivious sorting algorithm.
- Provable that no better cache-oblivious bound exists.

Performance: Serial Execution

- Actual implementation performed poorly!
- Runs significantly slower than Quicksort (about 4x as slow).

Why? Possible Reasons

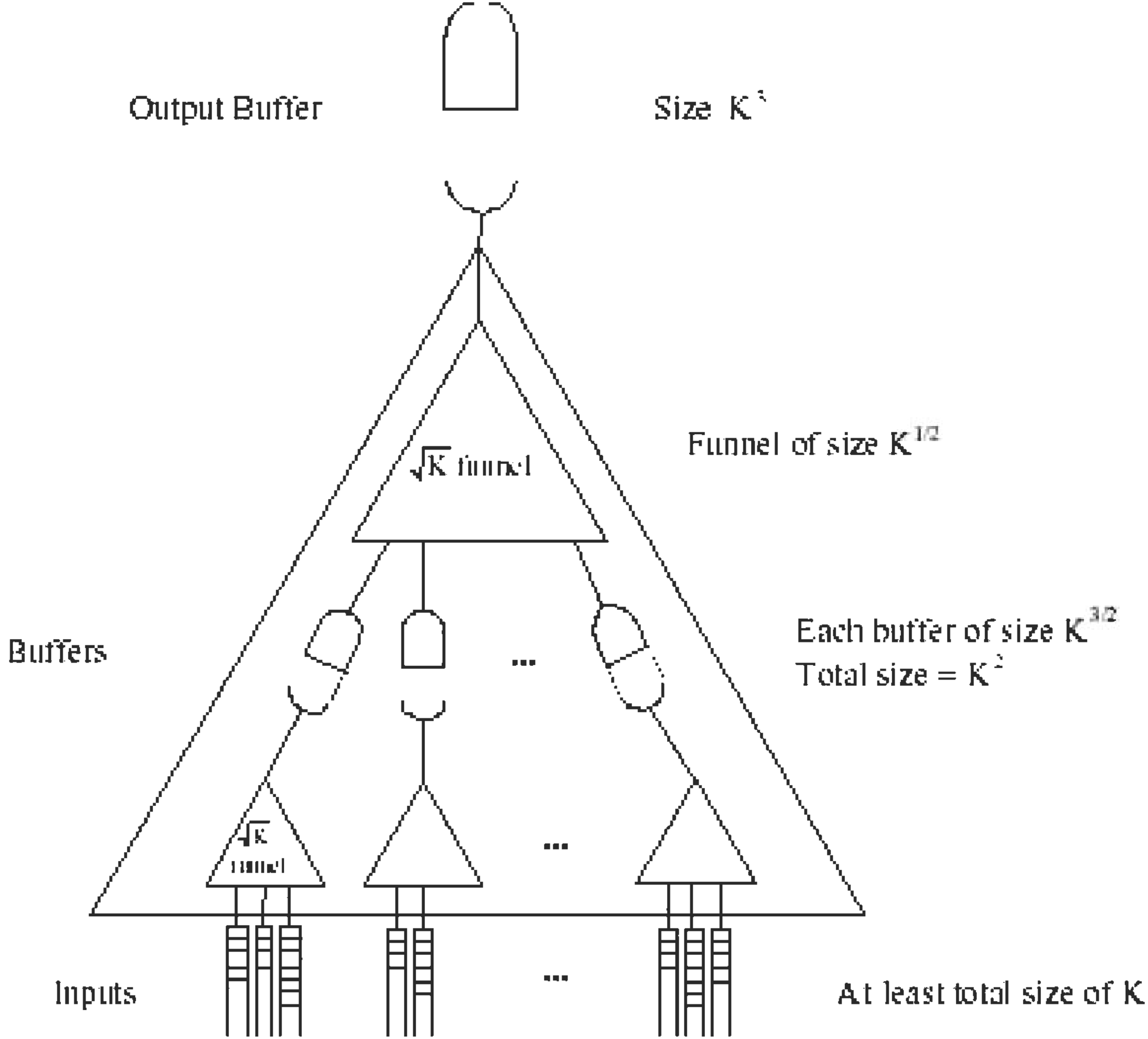
- Bad implementation.
- Runtime not dominated by cache misses.
- Much more memory management than Quick Sort (not in place).
- Lots of calculations to keep track of buffers and how full they are.
- Cache dominated by the internal buffers of a funnel, for a *K*-funnel, *K*^2 memory used for internal buffers, 2*K used for input buffers.
- Rounding errors! Funnel sort relies on taking square roots, and cube roots often, which in practice yields an imbalanced funnel.

Cache Performance

- Tested by changing type of item being sorted from int to long. Should approximately double the number of cache misses!
- Change in performance is less than 5% slower in a 4-way merge sort (Cilk Sort, implemented by Matteo Frigo), implying perhaps that cache misses are not a large cost.
- Appears that Funnel Sort suffers the smallest slowdown from increasing the size of data (versus Quick Sort and Cilk Sort), but difficult to say accurately.

Parallelize FunnelSort?

- Not terribly practical without a fast serial implementation.
- Lends well to parallelism.
  - Recursive merging can be done on separate processors.
  - Because of buffering, final merge shortly after inputs start being processed.
  - Could use locks on the circular buffers so that simultaneous reads from the head and writes to the tail are possible.
  - With $\log(n)/\log(Z)$ processors, possibly $O(n)$ sorting?


Output Buffer
Size K^3
√K funnel
Funnel of size K^1/2
Buffers
Each buffer of size K^3/2
Total size = K^2
√K funnel
Inputs
At least total size of K^3

Conclusions

- Possible to create a cache-oblivious sorting algorithm that has cache misses on the order of a cache-aware algorithm.
- In practice, difficult to implement correctly.
- Extra overhead difficult to recover with the reduced cache misses.
- Potentially very parallelizable.