Cache-Efficient Parallel Partition Algorithms

Alek Westover

MIT PRIMES

October 20, 2019

Partition

Definition (Partitioned Array)

$A[i]$ *predecessor*, $A[j]$ *successor* $\implies i < j$

Definition (Array partitioned relative to pivot value p)

$A[i] \leq p, A[j] > p \implies i < j$

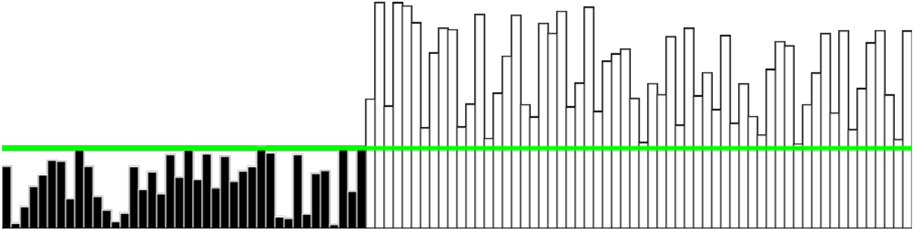
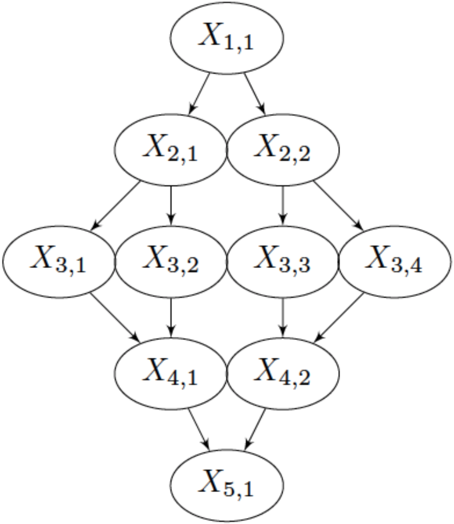


Figure: Black: Predecessor, White: Successor, Green: Pivot Value

Parallel Algorithm



Work and Span

Definition (T_p)

Running time on p processors. Note: $T_p \geq T_\infty$, $T_p \geq \frac{T_1}{p}$.

Definition (Work)

Running time on a single processor. $T_1 = \sum_i W_i$.

Definition (Span)

Running time on infinitely many processors. $T_\infty = \sum_i 1$.

Theorem (Brent's Theorem)

$$T_p = \sum_i \left\lceil \frac{W_i}{p} \right\rceil \leq \sum_i \left(\frac{W_i}{p} + 1 \right) = \frac{T_1}{p} + T_\infty.$$

Serial Partition

while low $<$ high **do**
    while $A[\text{low}] \leq$ pivotValue **do**
        low $\leftarrow$ low $+ 1$
    end while
    while $A[\text{high}] >$ pivotValue **do**
        high $\leftarrow$ high $- 1$
    end while
    Swap $A[\text{low}]$ with $A[\text{high}]$
end while
if $A[\text{low}] \leq$ pivotValue **then**
    low $\leftarrow$ low $+ 1$
end if

Randomized Algorithms

Definition (With high probability in n)

Probability of success is

$$1 - \frac{1}{n^c}$$

for c of our choice. i.e. the probability can be made arbitrarily close to 1.

Memory Bandwidth Bound

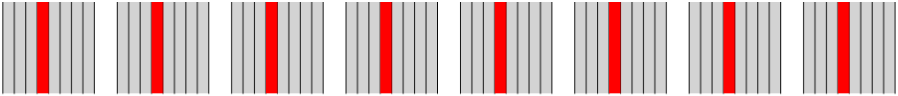
Definition (Cache miss)

A cache miss occurs when the algorithm must load a cache-line not stored in cache into cache.

- In-place $\implies$ spatial-locality
- Kuszmaul developed in-place parallel partition algorithm
  - Outperforms standard out-of-place algorithm
  - Outperformed by more cache-efficient higher span algorithm
- Experiments show Memory Bandwidth Bound (incurring too many cache misses) is the problem
- Want Temporal and Spatial locality

Strided Algorithm [1, 2] Description

Partition A into chunks $C_1, C_2, \ldots C_n/gb$ each consisting of g cache lines of size b. Let P_i be the union of the i-th cache-line from each chunk C_j.



Definition (Partially Partitioned Array)

$\exists u, l$ *such that*

$$i < u \implies A[i] \text{ is predecessor } , i \geq l \implies A[i] \text{ is successor}$$

- Perform serial partitions on all P_i in parallel.
- Let v_i be the position in A of the first successor in P_i. Perform a serial partition on $A[\min_i v_i], \ldots, A[\max_i v_i - 1]$.

Strided Algorithm Analysis

- Partial partiton step: work $O(n)$, span $\Theta(n/g)$.
- Serial cleanup step: span $\Theta(v_{\max} - v_{\min})$, which is $O(n)$ in general.
- If the number of predecessors in each P_i is similar, $v_{\max} - v_{\min}$ can be small.
- In particular, if $b \in \text{polylog}(n)$, and the array values are selected independently at random from some distribution, and g is chosen to optimize span ($g = n^{1/3}$), then with high probability in n,

  $$v_{\max} - v_{\min} < \tilde{O}(n^{2/3}),$$

  the span is

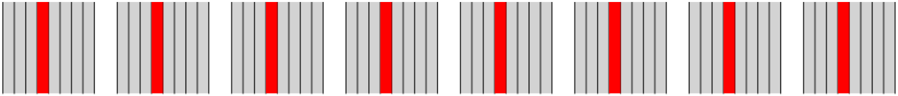
  $$\tilde{O}(n^{2/3}),$$

  and the number of cache misses is fewer than

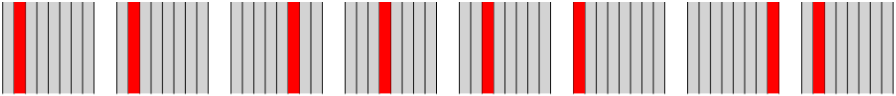
  $$\frac{n}{b} + \frac{\tilde{O}(n^{2/3})}{b}.$$

Blocked Strided Algorithm to Smoothed Striding Algorithm

Blocked Strided Algorithm P_i.

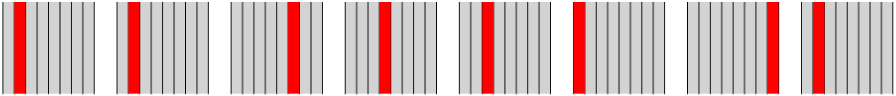


Smoothed-Striding Algorithm U_i.



Smoothed Striding Algorithm Description

Let $X[1], \ldots, X[s]$ be chosen uniformly at random from $\{1, \ldots, g\}$. Let U_i be the union of the $(X[j] + i) \mod g$-th cache-line from each chunk C_j.



- Perform serial partitions on all U_i in parallel.
- The array is partially now partitioned with $A[i]$ a predecessor for all $i < v_{\min}$ and $A[i]$ a successor for all $i \geq v_{\max}$.

Note that we will make $s = \frac{n}{gb} < \text{polylog}(n)$ so the algorithm remains in-place.

Partial Partition Step Analysis

Proposition

Let $\epsilon \in (0, 1/2)$ *and* $\delta \in (0, 1/2)$ *such that* $\epsilon \geq \frac{1}{\mathrm{poly}(n)}$ *and* $\delta \geq \frac{1}{\mathrm{polylog}(n)}$. *Suppose* $s > \frac{\ln(n/\epsilon)}{\delta^2}$. *Finally, suppose that each processor has a cache of size at least* $s + c$ *for a sufficiently large constant* c.
Then the Partial-Partition Algorithm achieves work $O(n)$*; achieves span* $O(b \cdot s)$*; incurs* $\frac{s+n}{b} + O(1)$ *cache misses; and guarantees with probability* $1 - \epsilon$ *that*

$$v_{max} - v_{min} < 4n\delta.$$

From Partial Partition to Full Partition

Partial Partition Step:

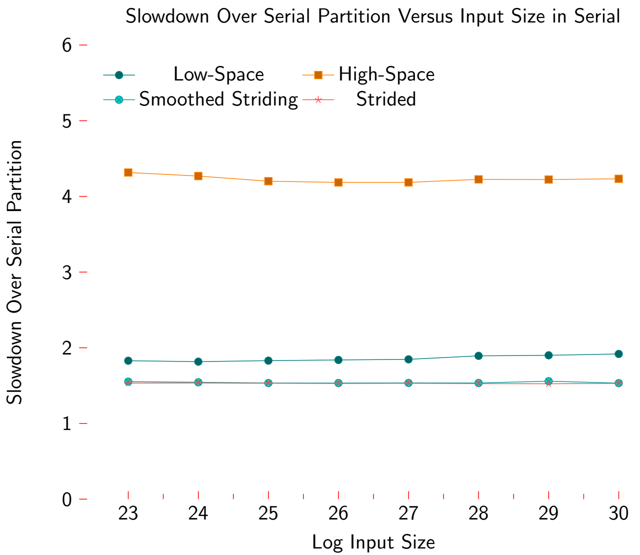
- Use $\epsilon = 1/n^c$ for c of our choice (i.e. with high probability).
- Choice of δ results in tradeoff between cache misses and span.

Recursive strategies:

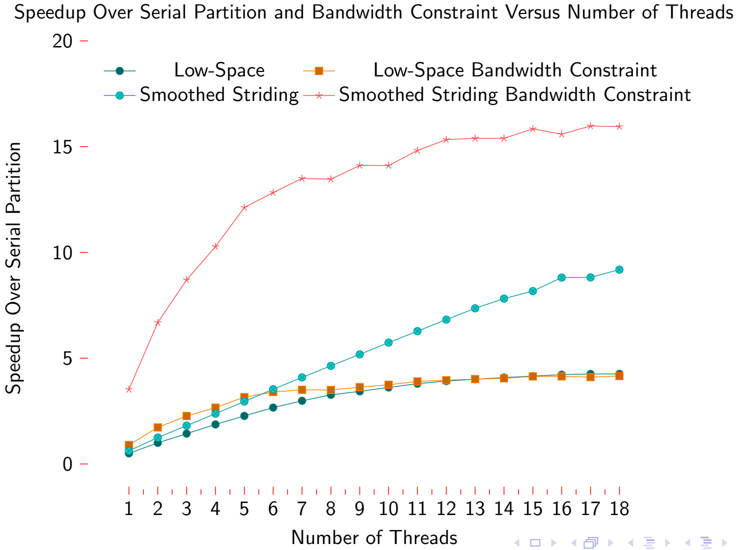
- **Hybrid Smoothed Striding Algorithm**: Use algorithm with span $O(\log n \log\log n)$. Note: recursive algorithm's cache behavior doesn't affect overall cache behavior because subarray is small. This algorithm can be tuned to give optimal span and cache misses.
- **Recursive Smoothed Striding Algorithm**: Use the Partial Partition step recursively to solve subproblems. Recursive applications of the Partial Partition step use the same ϵ the top-level (to guarantee success with high probability in n), and use $\delta \in \Theta(1)$ such that the problem size is reduced by half at each step. This algorithm has slightly worse span, but is very simple to implement.

Space Reduction (Spatial Locality) Yields Speedup

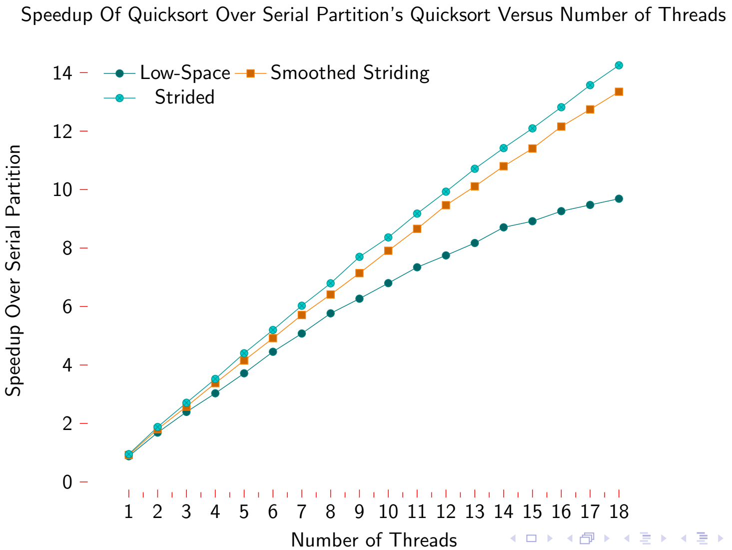
Space Reduction (Spatial Locality) Yields Speedup



Few passes over input (Temporal locality) reduces memory bandwidth bound



Smoothed-Striding algorithm is comparable to Blocked Strided algorithm



Acknowledgments

I would like to thank

- The MIT PRIMES program
- William Kuszmaul, my PRIMES mentor
- My parents

References

Rhys S. Francis and LJH Pannan.
A parallel partition for enhanced parallel quicksort.
Parallel Computing, 18(5):543–550, 1992.

Leonor Frias and Jordi Petit.
Parallel partition revisited.
In *International Workshop on Experimental and Efficient Algorithms*, pages 142–153. Springer, 2008.