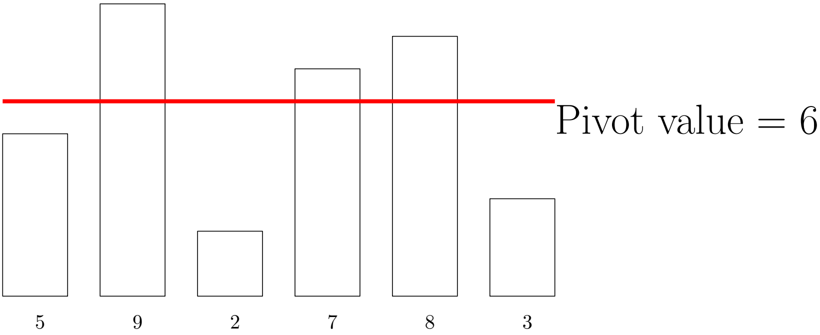
Cache-Efficient Parallel Partition Algorithms

Alek Westover

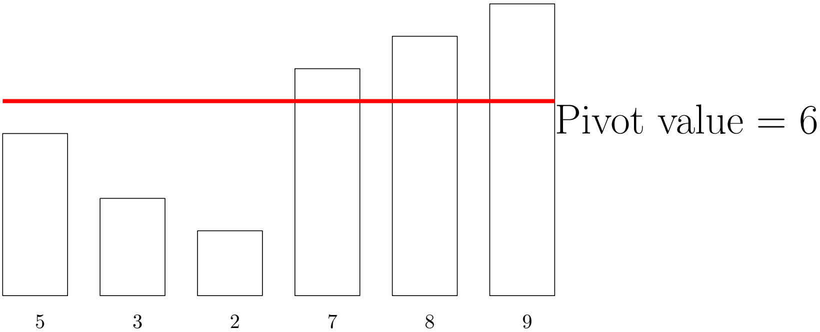
MIT PRIMES

October 20, 2019

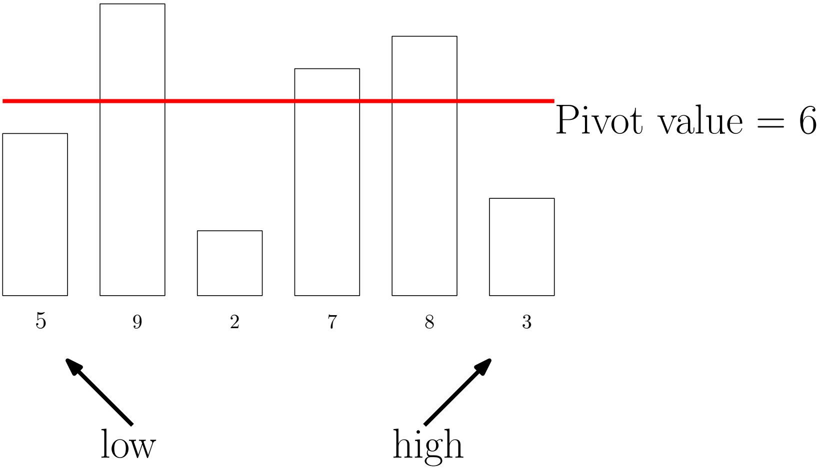
The Partition Problem



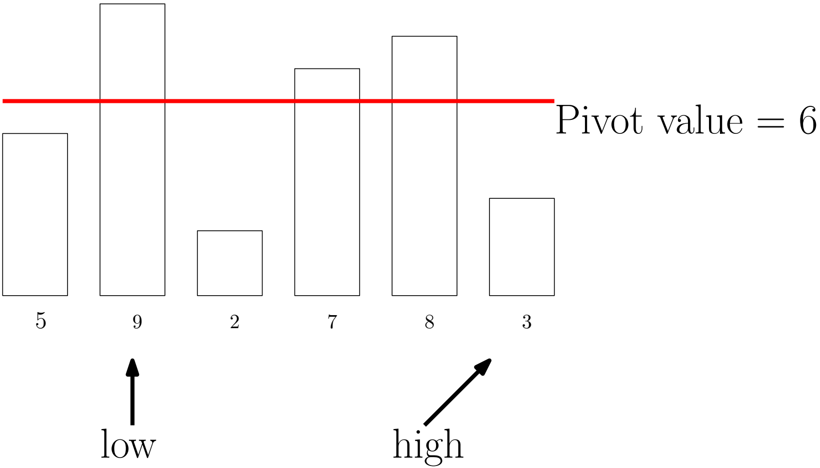
The Partition Problem



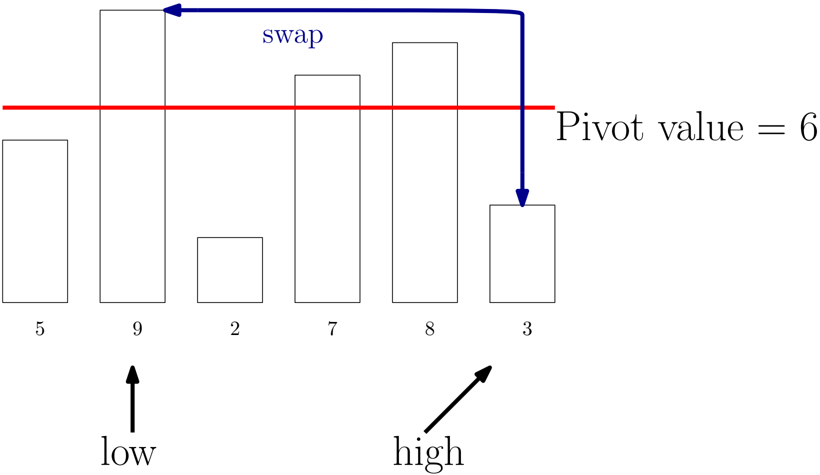
The Serial Partition Algorithm



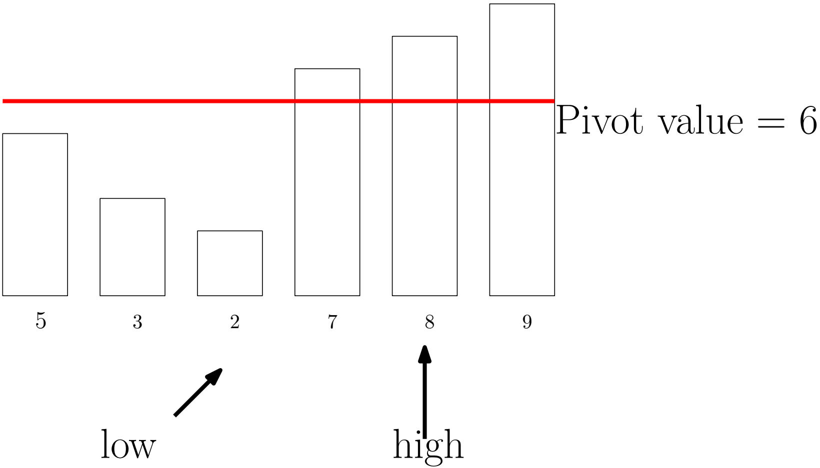
The Serial Partition Algorithm



The Serial Partition Algorithm

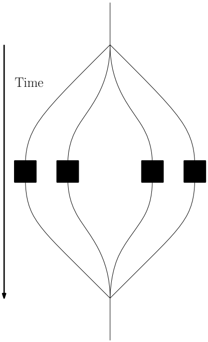


The Serial Partition Algorithm



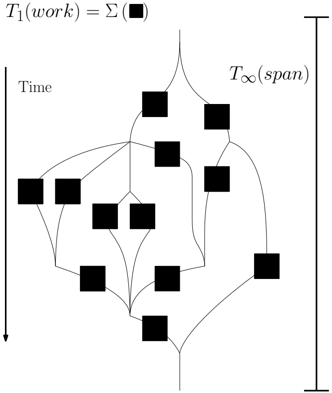
What is a Parallel Algorithm?

Parallel for loop



What is a Parallel Algorithm?

Composition



Bounding T_p with Work and Span

Brent's Theorem: [1]

$$T_p = \Theta\left(\frac{T_1}{p} + T_\infty\right)$$

[1] **[?]**

The Standard Parallel Partition Algorithm

Step	*Span*
Create filtered array	$O(1)$
Compute prefix sums of filtered array	$O(\log n)$
Use prefix sums to partition array	$O(1)$

Total span: $O(\log n)$

The Problem

- Standard algorithm is not in-place

THE PROBLEM

- Standard algorithm is not in-place
- Standard algorithm is not cache-efficient

THE PROBLEM

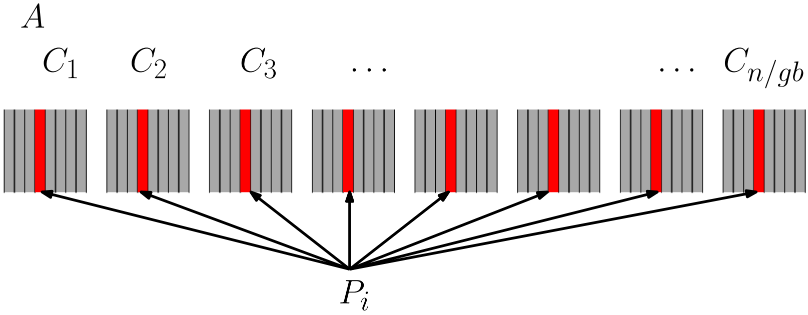
- Standard algorithm is not in-place
- Standard algorithm is not cache-efficient
- State-of-the-art solution lacks theoretical guarantees

The Problem

- Standard algorithm is not in-place
- Standard algorithm is not cache-efficient
- State-of-the-art solution lacks theoretical guarantees

Cache-Efficiency: (roughly)
An algorithm is (temporally) cache-efficient if it makes a single pass over the array.
This means that the algorithm will make relatively few requests to load data from memory into cache where it can be manipulated.

Strided Algorithm Description [2]



- Logically partition A_i into P_i so that P_i has the i-th block from each chunk C_j of the array
- 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 $A[\min_i v_i], \ldots, A[\max_i v_i - 1]$.

[2] [?, ?]

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.
- If array values are selected independently at random from some distribution, for appropriate choice of parameters, with high probability in n, the Strided Algorithm achieves span

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

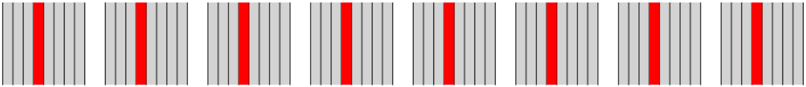
  and the number of cache misses is fewer than

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

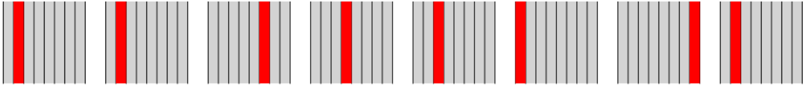
Our Result: The Smoothed Striding Algorithm

By randomly perturbing the internal groupings of the Strided Algorithm we remove the Strided algorithm's need for random inputs and get an algorithm with theoretical guarantees for arbitrary inputs.

Blocked Strided Algorithm P_i.

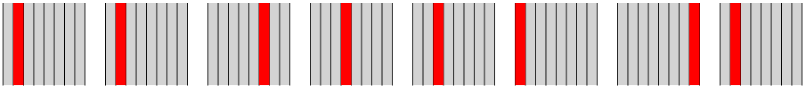


Smoothed-Striding Algorithm U_i.



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 $i < v_{\text{min}}$ and $A[i]$ a successor for all $i \geq v_{\text{max}}$.

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:

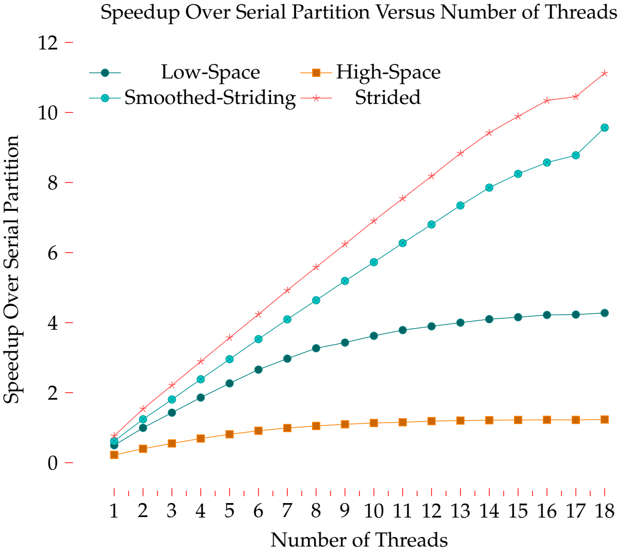
- Use $\epsilon = 1/n^c$ for c of our choice (i.e. with high probability).
- Choice of δ results in tradeoff between cache misses and span.

From Partial Partition to Full Partition

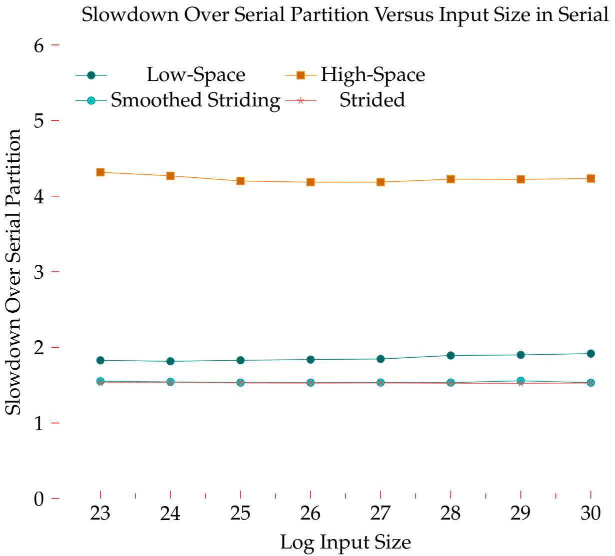
Recursive strategies:

- ***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.

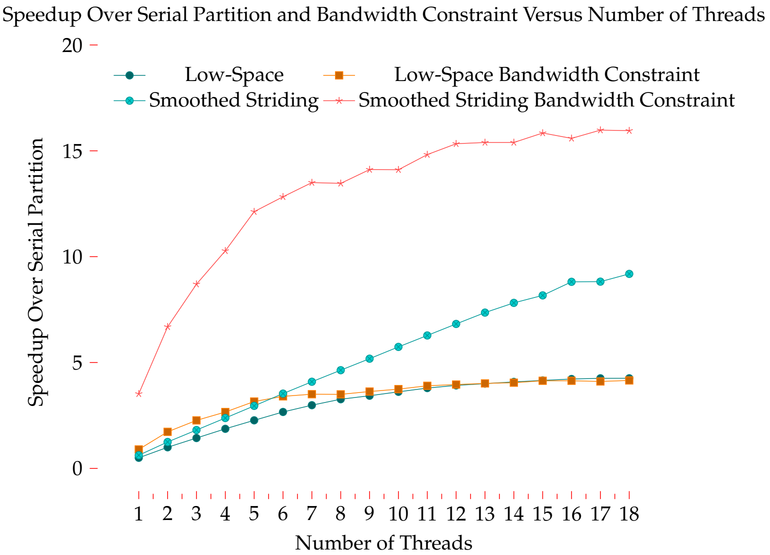
Space Reduction (Spatial Locality) Yields Speedup



Space Reduction (Spatial Locality) Yields Speedup

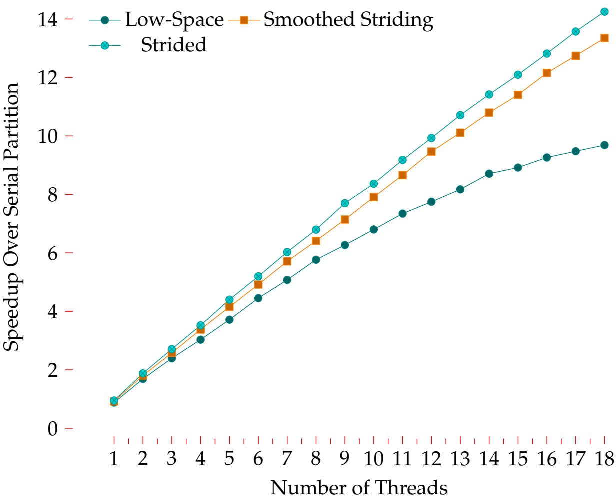


Few passes over input (Temporal locality) reduces memory bandwidth bound



Smoothed-Striding algorithm is comparable to Blocked Strided algorithm

Speedup Of Quicksort Over Serial Partition's Quicksort Versus Number of Threads



ACKNOWLEDGMENTS

I would like to thank

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

References

Richard P Brent.
The parallel evaluation of general arithmetic expressions.
Journal of the ACM (JACM), 21(2):201–206, 1974.

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.