Cache-Efficient Parallel Partition Algorithms Using Exclusive-Read-and-Write Memory

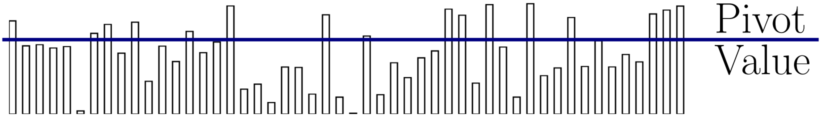
William Kuszmaul, Alek Westover

MIT

July 4, 2020

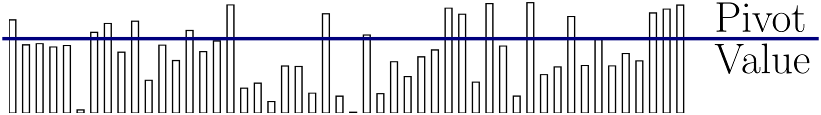
The Partition Problem

An unpartitioned array:

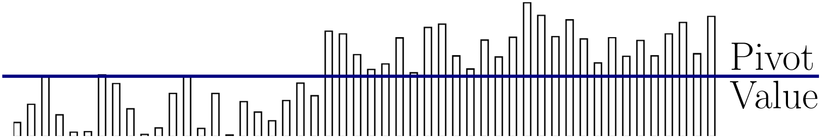


The Partition Problem

An unpartitioned array:



An array partitioned relative to a pivot value:

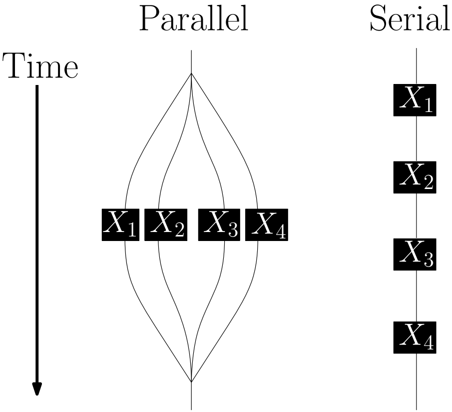


Example Applications

- Parallel Partition
- Parallel Quicksort
- Filtering Operations

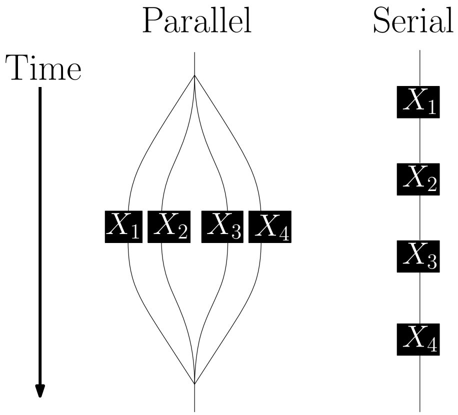
Our Model of Parallelism

Language based model. Primitive: ***Parallel for loop***.
Similar to how Cilk works



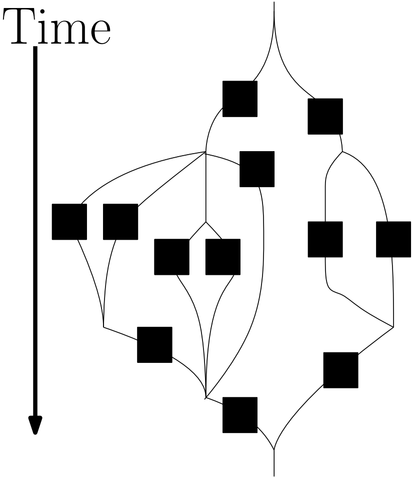
Our Model of Parallelism

Language based model. Primitive: ***Parallel for loop***.
Similar to how Cilk works



Rule for this talk: no locks or atomic variables

Parallel Algorithm Performance Metrics



T_p: Time to run on p processors

Work: T_1

- time to run in serial

Span: T_∞

- time to run on infinitely many processors

Brent's Theorem:

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

Optimizing Cache Efficiency

Cache-efficient algorithm: serial implementation incurs $(1 + o(1))n/B$ cache misses in the Disk Access model.

[Aggarwal and Vitter, 1988]

- Perform low number of passes over the data
- Don't use extra memory (*In-Place*)
- Simultaneously operate on elements close in memory

PREVIOUS WORK

"Standard Algorithm": span $O(\log n)$, but slow in practice

- Uses extra memory
- Makes multiple passes over data

$\left.\begin{array}{l} \\ \\ \end{array}\right\}$ bad cache behavior

Fastest algorithms in practice:

- Lock based and atomic-variable based algorithms

  [Michael Axtmann, Sascha Witt, Daniel Ferizovic, and Peter Sanders, 2017; Philip Heidelberger, Alan Norton, and John T. Robinson, 1990; Philippas Tsigas and Yi Zhang, 2003]

- The Strided Algorithm

  [Francis and Pannan, 92; Frias and Petit, 08]

  No locks or atomic-variables, lacks theoretical guarantees

Our Question

Can we create an algorithm with *theoretical guarantees* that is *fast in practice*?

Our Result

The Smoothed-Striding Algorithm

Key Features:

- linear work and polylogarithmic span
  (like the Standard Algorithm)
- fast in practice
  (like the Strided Algorithm)
- theoretically optimal cache behavior
  (unlike any past algorithm)

STRIDED VERSUS SMOOTHED-STRIDING ALGORITHM

Strided Algorithm

[Francis and Pannan, 92; Frias and Petit, 08]

- Good cache behavior in practice
- Worst case span is $T_\infty \approx n$
- On random inputs span is $T_\infty = \tilde{O}(n^{2/3})$

Strided Versus Smoothed-Striding Algorithm

Strided Algorithm

[Francis and Pannan, 92; Frias and Petit, 08]

- Good cache behavior in practice
- Worst case span is $T_\infty \approx n$
- On random inputs span is $T_\infty = \tilde{O}(n^{2/3})$

Smoothed-Striding Algorithm

- Provably optimal cache behavior
- Span is $T_\infty = O(\log n \log \log n)$ with high probability in n
- Uses randomization *inside* the algorithm

Smoothed-Striding Algorithm's performance