External String Sorting: Faster and Cache-oblivious

Rolf Fagerberg
Universiy of Southern Denmark

Anna Pagh
IT University of Copenhagen

Rasmus Pagh
IT University of Copenhagen

STACS 2006, February 23, 2006

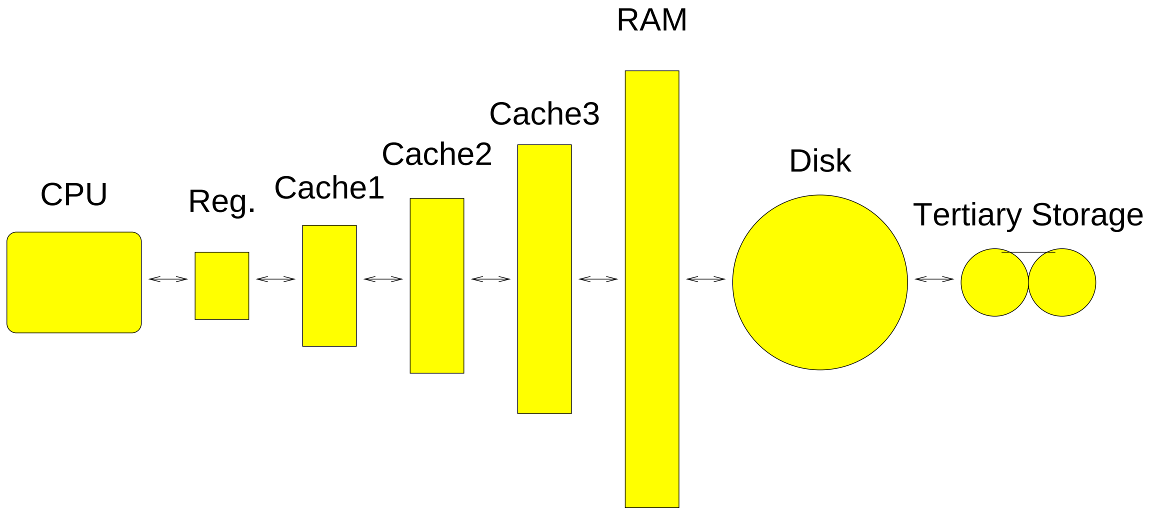
The Problem

Sorting Strings in External Memory

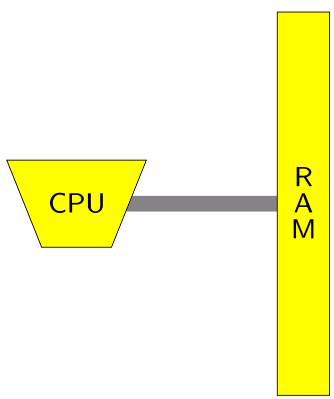
Strings:	Ubiquitous data type (word processing, DBs, WWW, bioinformatics, ...).
	Integers and multi-dimensional data are special cases.
External Memory:	Disks are slow, so minimize I/Os.

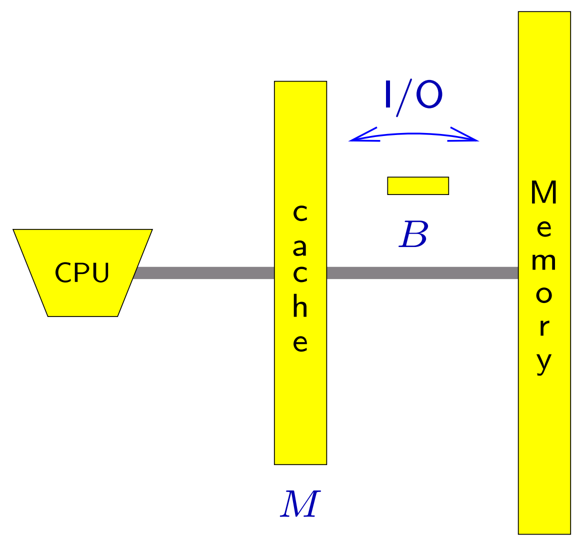
I/O model

Reality:



Models:





RAM model (cost: CPU time)

I/O model (cost: I/Os)

Existing Internal Bounds

Internally, string sorting is well solved:

$$\Theta(\mathrm{Sort}(K) + N)$$

where

K = # strings.

N = total # characters in strings.

$\mathrm{Sort}(K)$ = time to sort K elements of alphabet.

For comparison based alphabet: $\mathrm{Sort}(K) = \Theta(K \log K)$.

For integer alphabet (on word-RAM): $\mathrm{Sort}(K) = O(K\sqrt{\log\log K})$.

Existing External Bounds

Externally, we by analogy could hope to meet the lower bound

$$\Omega(\mathrm{Sort}(K) + N/B)$$

where $\mathrm{Sort}(K)$ = I/O cost to sort K elements = $\frac{K}{B}\log_{M/B}\frac{K}{M}$.

Best existing bound (slightly simplified): [Arge et al., STOC'97]

$$O(\frac{N_1}{K_1} \cdot \mathrm{Sort}(K_1) + B \cdot \mathrm{Sort}(K_2) + N/B)$$

	# strings	# characters
Short strings	K_1	N_1
Long strings	K_2	N_2

Short strings: at most B characters

Long strings: more than B characters

This Paper

New upper bound:

$$O(\mathrm{Sort}(K) \cdot \log\log_M(K) + N/B)$$

This Paper

New upper bound:

$$O(\mathrm{Sort}(K) \cdot \log \log_M (K) + N/B)$$

Goal:

$$O(\mathrm{Sort}(K) + N/B)$$

Existing upper bound:

$$O(\mathrm{Sort}(K) \cdot B + N/B)$$

This Paper

New upper bound:

$$O(\mathrm{Sort}(K) \cdot \log\log_M(K) + N/B)$$

Goal:

$$O(\mathrm{Sort}(K) + N/B)$$

Existing upper bound:

$$O(\mathrm{Sort}(K) \cdot B + N/B)$$

$B = 10^3$, $M = 10^6$:

$$\log\log_M(K) \geq B$$

$$K \geq 10^{6 \cdot 2^{(10^3)}} \approx 10^{(10^{302})}$$

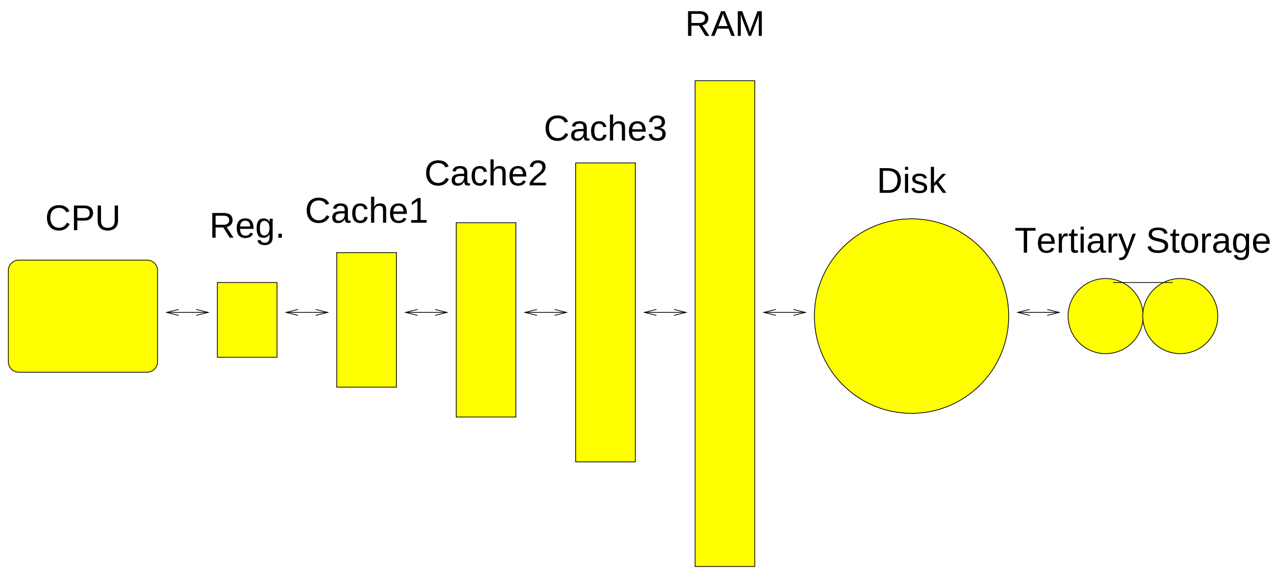
This Paper

Our algorithm:

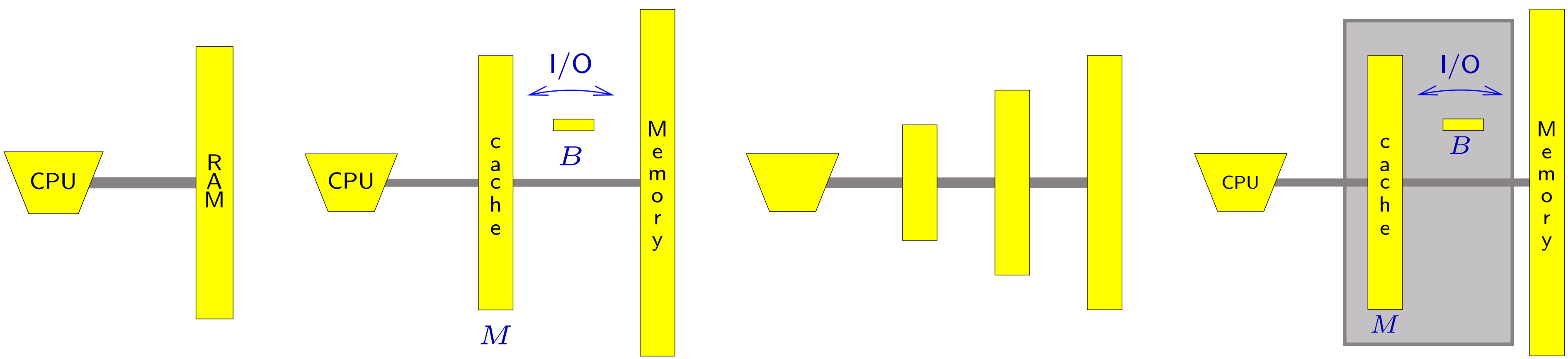
- Is randomized, with error bound $O(1/N^c)$ for any c (at a price of constant factor c in complexity bound).
- Finds sorted order ("rank-sorting") and LCP array of input strings (no permutation of the strings).
- Works in the cache-oblivious model.

Cache-Oblivious Model

Reality:



Models:



RAM model

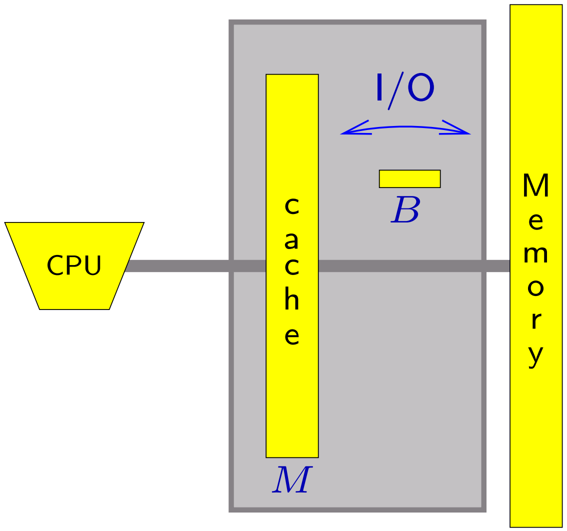
I/O model

Multi-level models

Cache-oblivious model

Cache-Oblivious Model

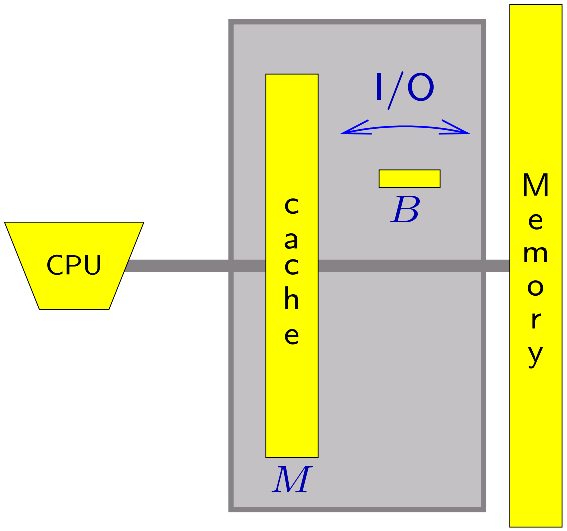
- Program in the RAM model
- Analyze in the I/O model for arbitrary B and M
- Optimal off-line cache replacement strategy



[Frigo, Leiserson, Prokop, Ramachandran, FOCS'99]

Cache-Oblivious Model

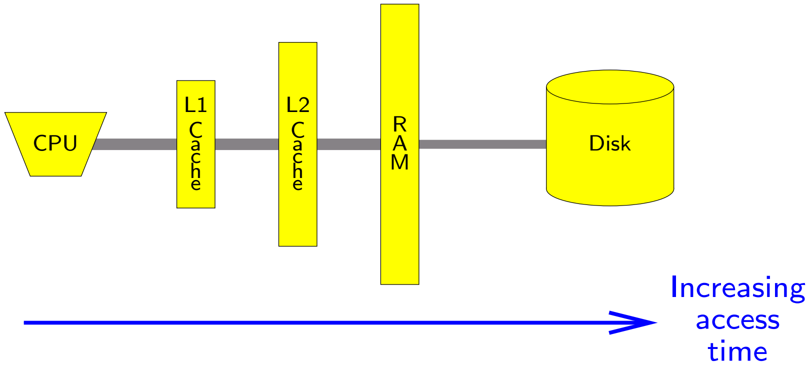
- Program in the RAM model
- Analyze in the I/O model for arbitrary B and M
- Optimal off-line cache replacement strategy



[Frigo, Leiserson, Prokop, Ramachandran, FOCS'99]

Advantages:

- Optimal on arbitrary level $\Rightarrow$ optimal on all levels
- Simplicity of model.
- Portability
- Robustness in multiprocess systems



Cache-Oblivious String Sorting

Best existing result:

$$O(\mathrm{Sort}(N))$$

by reduction to suffix tree construction algorithm

[Farach, FOCS'97]
[Farach-Colton et al., JACM 00]

New upper bound works in cache-oblivious model:

$$O(\mathrm{Sort}(K) \cdot \log \log_M(K) + N/B)$$

Rank Sorting of Strings

Output:

- Array of pointers to strings in sorted order (rank array).
- Array of lengths of Longest Common Prefix (as well as branching chars) of neighboring strings in the sorted order (LCP array).

Cf. suffix arrays.

The essential information in a compressed trie over the strings.

Important Application

Improved construction of External String Dictionaries.

I/O-Model [Ferragina, Grossi, STOC'95]

Cache-Oblivious Model [Brodal, Fagerberg, SODA'06]

Searching for pattern P in these takes $O(\log_B K + |P|/B)$ I/Os, which is optimal.

Building these is equivalent to rank sorting (rank array + LCP array) in the respective models. Hence, our improvements carry over.

Algorithm

Input is binary strings (measured in words of $\log N$ bits).

Idea 1:

Repeatedly halve string lengths using hashing.

$$\ldots|\overset{a}{110100}|\overset{b}{010110}|\ldots \quad \longrightarrow \quad \ldots|\overset{h(ab)}{100011}|\ldots$$

Idea 2:

Find unordered compressed trie recursively, then make ordered at the end.

Inspiration: Word-RAM "signature sort" of Andersson et al. [STOC'95]

Algorithm

Input is binary strings (measured in words of $\log N$ bits).

Idea 1:

Repeatedly halve string lengths using hashing.

$$\ldots|\overset{a}{110100}|\overset{b}{010110}|\ldots \quad \longrightarrow \quad \ldots|\underbrace{\overset{h(ab)}{100011}}_{(c+2)\log N}|\ldots$$

$$P(\text{no collisions}) \le \binom{N}{2} \cdot 1/2^{(c+2)\log N} \le 1/N^c$$

Idea 2:

Find unordered compressed trie recursively, then make ordered at the end.

Inspiration: Word-RAM “signature sort” of Andersson et al. [STOC’95]

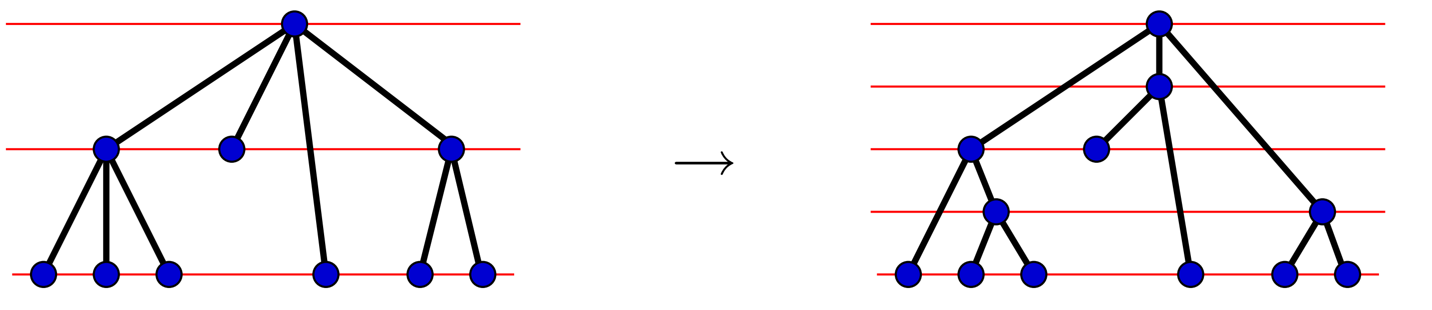
Algorithm

Relationship between LCP for strings and halved strings (assuming no collisions):

|110|010|011|110| $\longrightarrow$ |111|101|
|110|010|011|101| $\qquad$ |111|010|

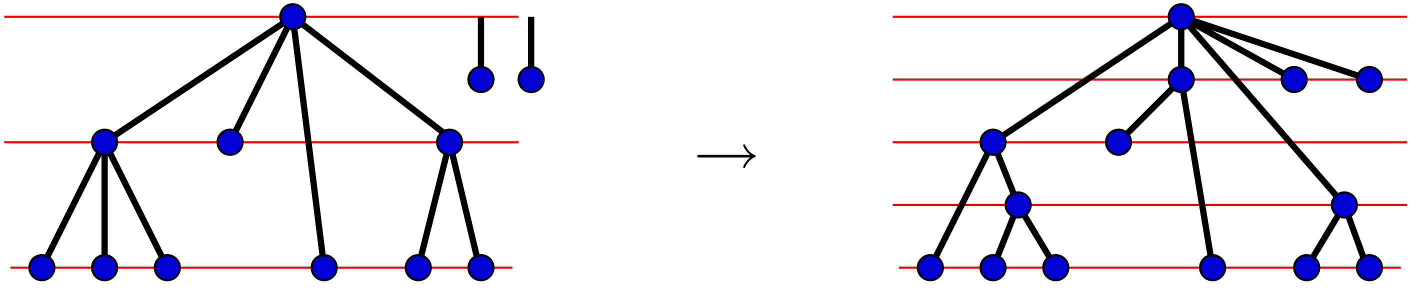
S_i = strings after i halving steps.

Construct unordered compressed trie for S_i from unordered trie for S_{i+1}:



Algorithm

Construct unordered compressed trie for S_i from unordered trie for S_{i+1}. Don't recurse on strings of length one.



In tries, keep only branching nodes and branching characters (hash values). At most $2 \cdot (\text{\# strings})$ nodes.

Expansion step: batched collecting of (pairs of) branching chars from halving level i (using sorting as rearrangement routine).

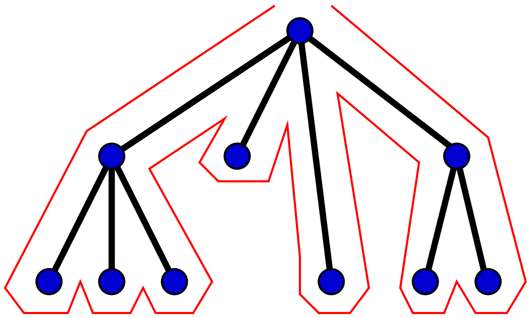
Takes $O(\mathrm{Sort}(\text{\# strings}) + (\text{\# chars})/B)$ I/Os.

Algorithm

Finally, make tree ordered (batched collecting of branching chars from actual strings, using sorting).

Takes $O(\mathrm{Sort}(K) + N/B)$ I/Os.

Create rank array and LCP array by creating Euler tour of tree, list ranking it, and traversing it.



Takes $O(\mathrm{Sort}(K))$ I/Os using existing Euler tour and list ranking algorithms.

Analysis

Space: Geometrically decreasing, $O(N)$ words in total.

Recursion (compression of strings/expansion of tries):

> Scanning during compression is geometrically decreasing, $O(N/B)$ I/Os in total.
>
> Expansion: $O(\mathrm{Sort}(\text{\#strings}))$ plus scanning per recursive level. No recursion on strings of length one. Hence, #strings $\leq$ #chars. Hence, after $\log \frac{N}{K}$ recursive levels, remaining levels cost $O(\mathrm{Sort}(K))$ in total due to geometrical decrease.

$$O(\mathrm{Sort}(K) \cdot \log(\frac{N}{K}) + N/B)$$

$$= O(\mathrm{Sort}(K) \cdot \log \log_M(K) + N/B)$$

Summary

New randomized algorithm for (rank-)sorting of strings in external memory.

Improves on existing deterministic ones, and is very close to the goal of $O(\mathrm{Sort}(K) + N/B)$.

I/O-Model:

Old: $O(\mathrm{Sort}(K) \cdot B + N/B)$

New: $O(\mathrm{Sort}(K) \cdot \log\log_M(K) + N/B)$

Cache-Oblivious Model:

Old: $O(\mathrm{Sort}(N))$

New: $O(\mathrm{Sort}(K) \cdot \log\log_M(K) + N/B)$