Parallel Sorting

Paradigms for Parallel Computing

João Luís Sobral

21/April/2015

Parallel Sorting

- Sequential sorting algorithms

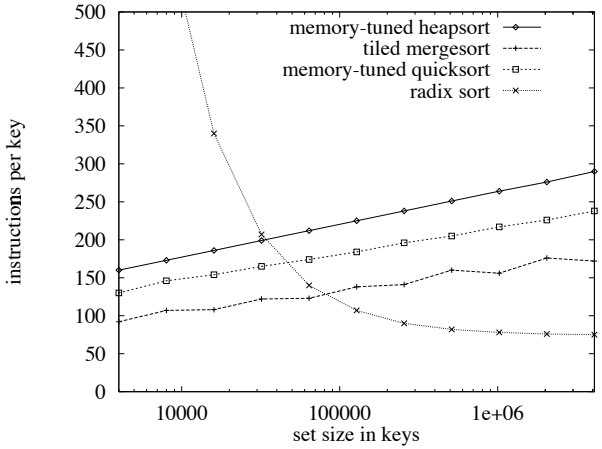
Method	Complexity (average)	Type	Description
Quicksort	*n* log *n*	Partitioning	Recursively sort elements less/ greater than a given pivot
Mergesort	*n* log *n*	Merging	Successively merge sorted sub-lists starting from lists with one element
Heap sort	*n* log *n*	Selection	Insert elements into a binary heap
Insertion sort	n^2	Insertion	Insert elements into the sorted list
Radix sort	n d		Sort elements digit by digit (d)

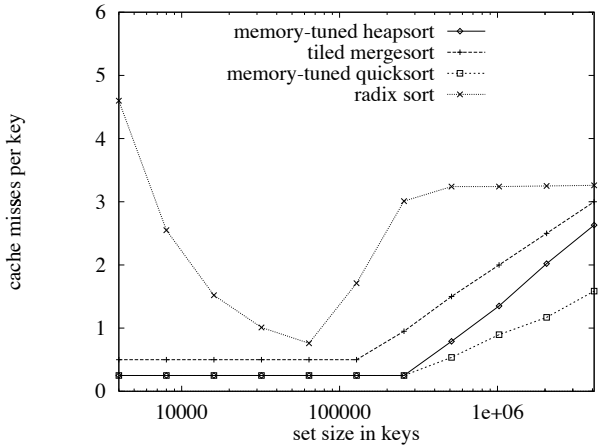
Parallel Sorting

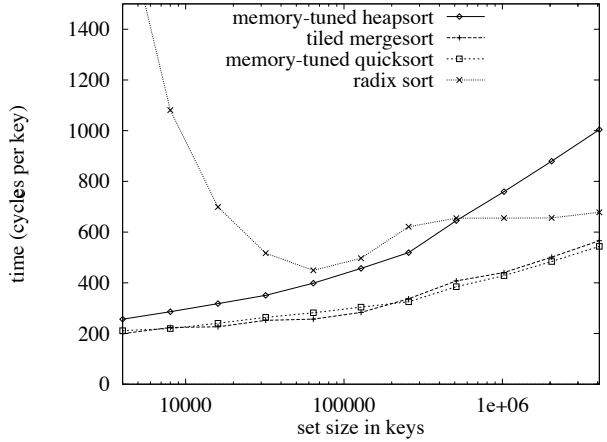
- Locality of reference in sorting algorithms

Method	Locality of reference	Improvements
Quicksort	Good spatial locality + bad temporal locality on initial stages	Initial set partioning using k keys
Mergesort	Good spatial locality + bad temporal locality on final merge stages	Single merge when data exceeds cache size
Heap sort	Bad	Cache aware trees + d-fan-out
Insertion sort	Bad	
Radix sort	Good when MSD first (only when processing LSDs)	Reduce the number of passes through data

Parallel Sorting (Impact on locality)







Parallel Sorting

- Parallelism in sorting algorithms

Method	
Quicksort	Start with p lists
Mergesort	Merge p lists parallel
Heap sort	???
Insertion sort	???
Radix sort	Sort set of digits in parallel

Parallel Sorting (on distributed memory)

- Design issues:
  - Keys are initially distributed over processors
  - Data properties
    - Partially-sorted data?
  - Exploitable parallelism
    - Splitter-based
    - Merger-based
  - Data movements across processors
  - Load balancing

Parallel Sorting

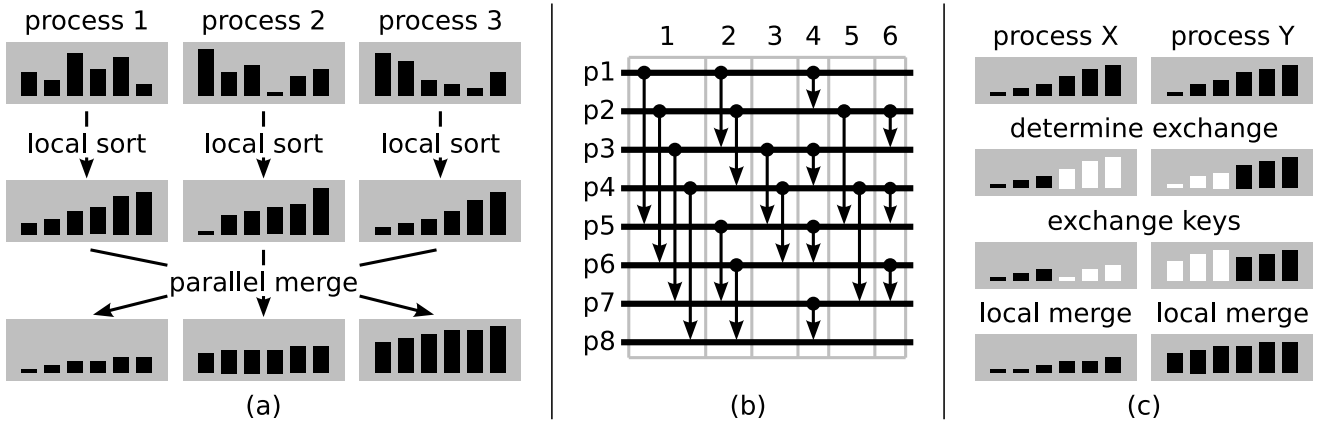
- Parallel quicksort (simplified)
  - Master selects and broadcasts *pivot* key
  - Each process locally splits using the *pivot*
    - Records size of *smaller* and *greater* sets
  - Sums size of *smaller* and *greater* sets
  - Divide processors into *smaller* and *greater* sets
    - Send data to each processor
  - Repeat the processes until #sets = #p
    - Locally sort on each process p

Parallel Sorting

- Parallelism in sorting algorithms
  - Mergesort
    - Merge data between pairs of processors (sorting networks)
    - Only effective when n/p ~1
    - Requires extensive data movements when n/p>>1
  - Sampling based
    - Split data into P sets using p-1 splitters
    - Each processor acts upon a local set
    - Minimizes data movements

Parallel Sorting

- Parallel mergesort
  - Locally sort each set
  - Exchange sets among processors

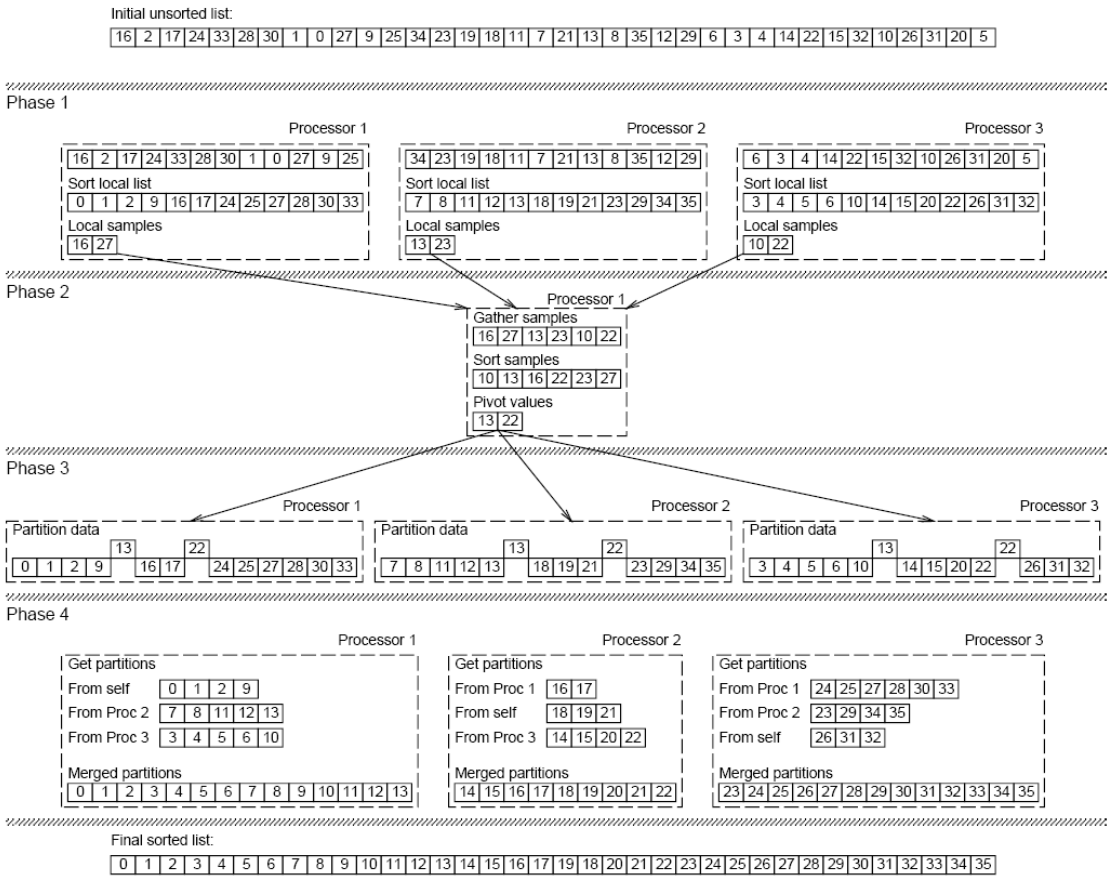


Parallel Sorting

- Sample sort
  - Quicksort based
    - Split data into P sets
    - Each processor acts upon a local set
    - Minimizes data movements
  - Regular sampling (p*(p-1) keys)
    - Not effective for large p
  - Random sampling
  - Histogram sampling

Parallel Sorting by Regular Sampling

1. Divide the set into *p* disjoint sets and locally order each set
   - Applies a local QuickSort
   - Selects *p-1* local samples that uniformly divide each set into p subsets
2. Order *p*(p-1)* samples and select best *p-1* pivot keys
3. Partition each set using the p-1 pivot keys
4. Merge *p*p* sets
   - Processor *i* merges the *i* partition



Parallel Sorting

- Parallel radix sort
  - Each processor is responsible by a subset of digit values
  - Sort and count the number of digit values
  - All-reduce the total number of digits
  - Send keys to the processor responsible for each digit range
  - Repeat for the next digit