PARADIS: AN EFFICIENT PARALLEL ALGORITHM FOR IN-PLACE RADIX SORT

M. Cho, D. Brand, R. Bordawekar, U. Finkler, V. Kulandaisamy, and R. Puri

Presented by: Helen He

Motivation

- Distribution based sorts achieve O(N)
- In-memory sorting due to I/O bounds on disk
- In-place sorting highly desirable
  - *Large in-memory databases*
  - *Fewer cache misses*
- Parallelizing in-place radix sort has been difficult due to read-write dependencies

MSD Radix Sort

Build a histogram of radix key distribution

Set pointers for input array distribution

Check elements and permute them if currently occupying wrong bucket

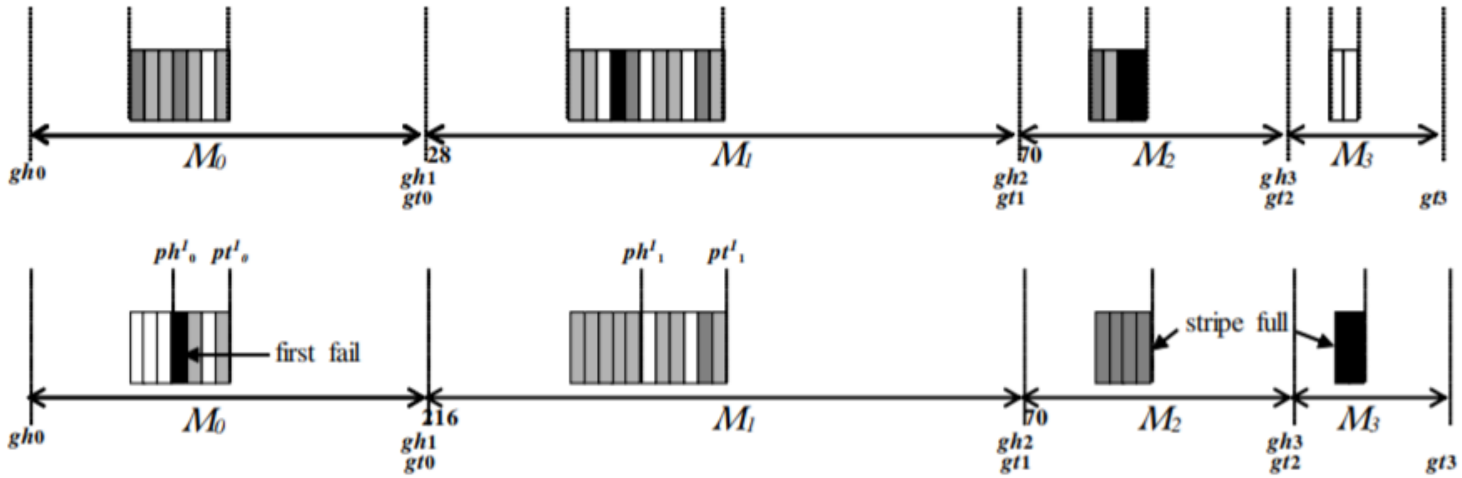
Recurse into subproblems for next digits

Algorithm 1 Radix Sort

```
procedure RadixSort(d[N],l)
    b = b_l                                  ▷ Function giving bucket at level l
    B = the range of b()
    cnt[B] = 0                               ▷ Histogram of bucket sizes
    for n ∈ N do
        cnt[b(d[n])]++
    end for
    for i ∈ B do
        gh_i = Σ_{j<i} cnt[j]
        gt_i = Σ_{j≤i} cnt[j]
    end for
    for i ∈ B do
        while gh_i < gt_i do                 ▷ Till bucket i is empty
            v = d[gh_i]
            while b(v)! = i do
                swap(v, d[gh_{b(v)}++])
            end while
            d[gh_i++] = v
        end while
    end for
    if l < L − 1 then                        ▷ Recurse on each bucket
        for i ∈ B do
            RadixSort(d[M_i],l+1)
        end for
    end if
end procedure
```

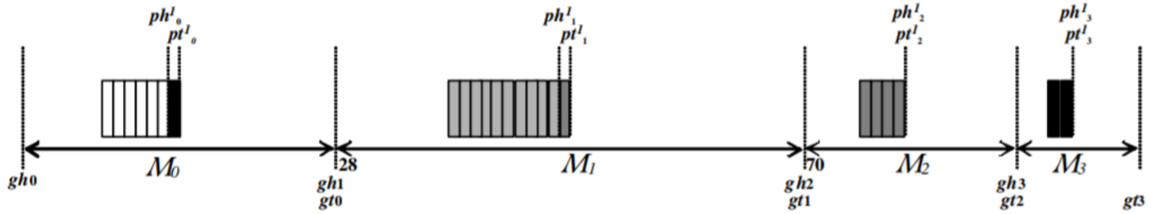
Speculative Permutation

- Need to partition work among P processors
- Ensuring the partitions are exact is difficult and expensive



- Aim to minimize the wrong bucket sizing and evenly split work among processors
- Each bucket split into |P| "stripes" -> each processor owns a stripe of each bucket

Speculative Permutation



Serial radix sort permutation

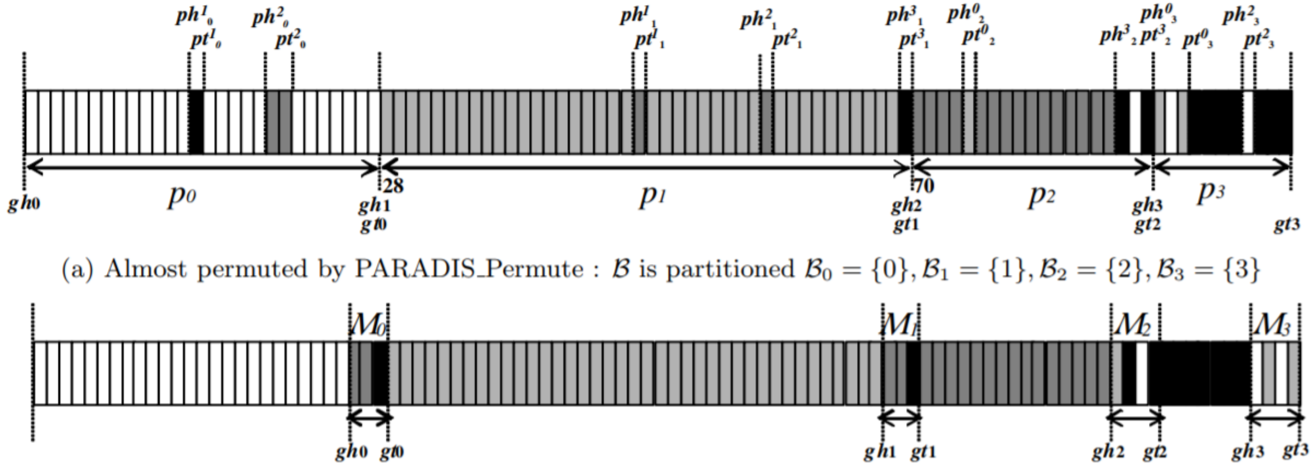
Move head pointer only if a correct element was found

Algorithm 3 PARADIS_Permute

procedure **PARADIS_Permute**(p)
  for $i \in \mathcal{B}$ **do**
    $head = ph_i^p$
    while $head < pt_i^p$ **do**
      $v = d[head]$ ▷ Keep moving v
      $k = b(v)$ ▷ to its bucket k
      while $k != i$ **and** $ph_k^p < pt_k^p$ **do**
        $swap(v, d[ph_k^p{+}{+}])$ ▷ v into its bucket k
        $k = b(v)$ ▷ New v and k
      end while
      if $k == i$ **then** ▷ Found a correct element
        $d[head{+}{+}] = d[ph_i^p]$
        $d[ph_i^p{+}{+}] = v$
      else
        $d[head{+}{+}] = v$
      end if
    end while
  end for
end procedure

Repair

- Partition the existing set of buckets B into disjoint subsets $B_p \subset B$, one for each processor $p \in P$



(a) Almost permuted by PARADIS_Permute : $\mathcal{B}$ is partitioned $\mathcal{B}_0 = \{0\}, \mathcal{B}_1 = \{1\}, \mathcal{B}_2 = \{2\}, \mathcal{B}_3 = \{3\}$

(b) Wrong elements moved to the end of buckets by PARADIS_Repair : $gh_{\{0,1,2,3\}}$ adjusted to the first wrong elements

- After repair we have a subproblem which we again run Permute and Repair on – opportunity for coarsening?

Load Balancing

- If there is a bucket which has way more elements than other buckets, this bucket will become the performance bottleneck
- PARADIS assigns each bucket *i* to a non-empty subset $P_i \subset P$. For any two buckets *i* and *j*, either $P_i = P_j$, or $P_i \cap P_j = \emptyset$.
  - Multiple processors can work on the same group of buckets, unlike Repair

$$\text{min: } max\{W(p) \mid \forall p\}$$
$$\text{where: } W(p) = \sum_{i \in \mathcal{B}_p} \frac{C_i \cdot log_{|\mathcal{B}|} C_i}{|\mathcal{P}_i|}$$

$$|\mathcal{P}_i| = |\mathcal{P}| \frac{C_i \cdot log_{|\mathcal{B}|} C_i}{\sum_{j \in \mathcal{B}} C_j \cdot log_{|\mathcal{B}|} C_j}$$

Estimation of P_i

  - Assign processors based on rounded $|P_i|$

Complexity Analysis

Lemma 1

- Let r_i be the ratio of wrong elements in bucket i over $|N|$
- Let E_i be the set of processors with an “empty” stripe for bucket i
- Let e_i be the ratio of E_i over all processors

$$r_i = \frac{C_i - C_i(i)}{|\mathcal{N}|} \leq \frac{C_i}{|\mathcal{N}|}(1 - e_i)$$

- $e_i C_i \leq C_i(i)$, because $e_i C_i$ represents the number of elements permuted into bucket i by processors in E_i

Lemma 2: $$r_i \leq e_i(1 - \frac{C_i}{|\mathcal{N}|}), \forall i$$

- Consider any other bucket j. In bucket i, any stripe p not in E_i still has the capacity to receive elements
- Any of these stripes p must have successfully permuted from bucket j any elements d[n] which satisfy b(d[n]) = i and are in a stripe of the same processor ($n \in M^p_j$)
- Therefore in bucket j, any element still belonging to i must be in a stripe $p \in E_i$

$$C_i = \sum_j C_j(i) \quad (1) \text{ and } C_i + \sum_{j \neq i} C_j = |\mathcal{N}|$$

$$C_j(i) = \sum_P C_j^p(i) = \sum_{E_i} C_j^p(i)$$

$$\leq \sum_{E_i} \frac{C_j}{|\mathcal{P}|} = \frac{|E_i|}{|\mathcal{P}|} C_j = e_i($$

$$r_i = \frac{C_i - C_i(i)}{|\mathcal{N}|} = \frac{\sum_{j \neq i} C_j(i)}{|\mathcal{N}|}$$

$$\leq e_i \frac{\sum_{j \neq i} C_j}{|\mathcal{N}|} = e_i(1 - \frac{C_i}{|\mathcal{N}|})$$

Bound on Ratio of Incorrect Keys

- Combining Lemmas 1 and 2, r_i is the min of both lemmas in the form "*min(x, y) – xy*" which is minimized at *x = y = .25*

$$r_i \leq min(\frac{C_i}{|\mathcal{N}|}(1 - e_i), e_i(1 - \frac{C_i}{|\mathcal{N}|})) \tag{14}$$

$$= min(\frac{C_i}{|\mathcal{N}|}, e_i) - e_i \frac{C_i}{|\mathcal{N}|} \leq \frac{1}{4} \tag{15}$$

Corollary 1

$$r = \sum_i r_i \leq \sum_i \frac{C_i}{|\mathcal{N}|} - \sum_i (\frac{C_i}{|\mathcal{N}|})^2 \tag{16}$$

which will be maximal with $C_i = \frac{|\mathcal{N}|}{|\mathcal{B}|}, \forall i$. *Thus*

$$r \leq 1 - \frac{1}{|\mathcal{B}|} \tag{17}$$

- Let w be the maximum fraction of wrong elements to be repaired, or $max\{ \sum_{i \in Bp} r_i \mid \forall p \}$

Theorem 2: $T(\mathcal{N}) \leq O(|\mathcal{N}|(\frac{1}{|\mathcal{P}|} + w))$

PROOF. *Without loss of generality, we let r and w represent their maxima over all iterations. Then*

$$T(\mathcal{N}) \leq (\frac{|\mathcal{N}|}{|\mathcal{P}|} + w|\mathcal{N}|) + r(\frac{|\mathcal{N}|}{|\mathcal{P}|} + w|\mathcal{N}|) + r^2(..) + ... \quad (18)$$

$$= \sum_{t=0}^{\infty} r^t(\frac{|\mathcal{N}|}{|\mathcal{P}|} + w|\mathcal{N}|) = (\frac{|\mathcal{N}|}{|\mathcal{P}|} + w|\mathcal{N}|)\frac{1}{1-r} \quad (19)$$

By Corollary 1, $\frac{1}{1-r} \leq |\mathcal{B}|$ which is constant. Hence

$$T(\mathcal{N}) \leq O(|\mathcal{N}|(\frac{1}{|\mathcal{P}|} + w)) \quad (20)$$

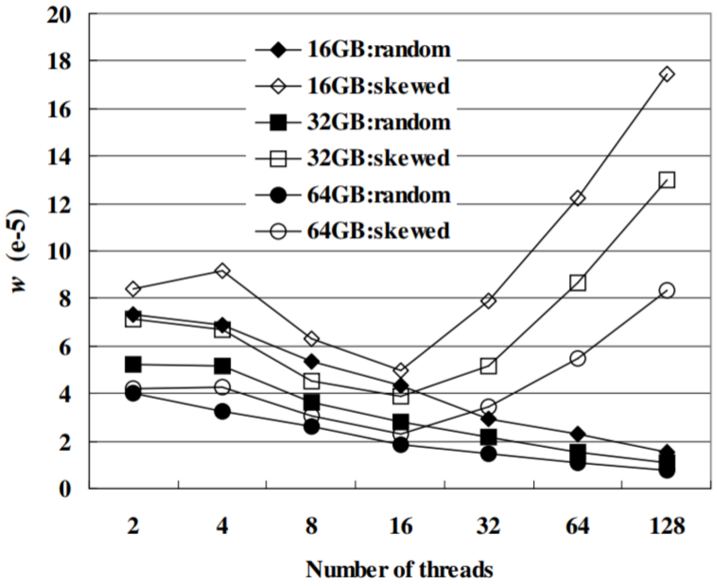
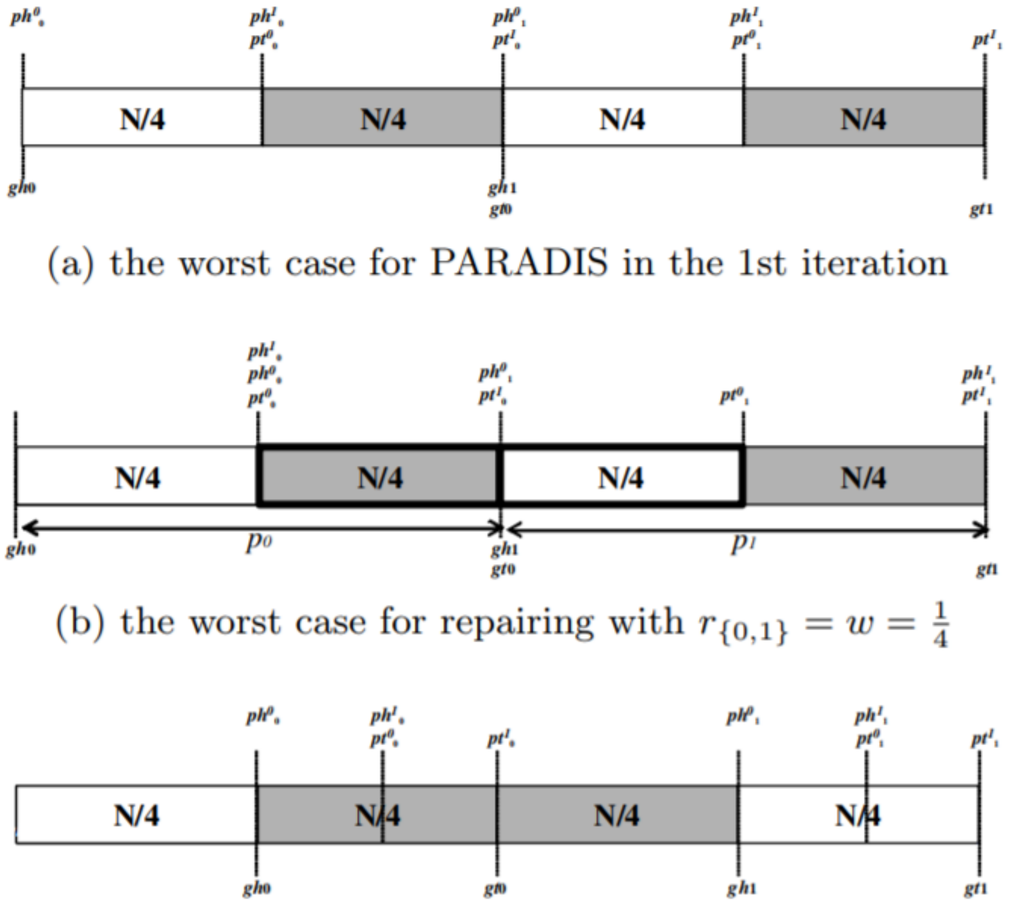


Figure 8: w **values from numeric benchmarks**

$T(\mathcal{N})$ converges to $O(\frac{|\mathcal{N}|}{|\mathcal{P}|})$, as w goes to 0.



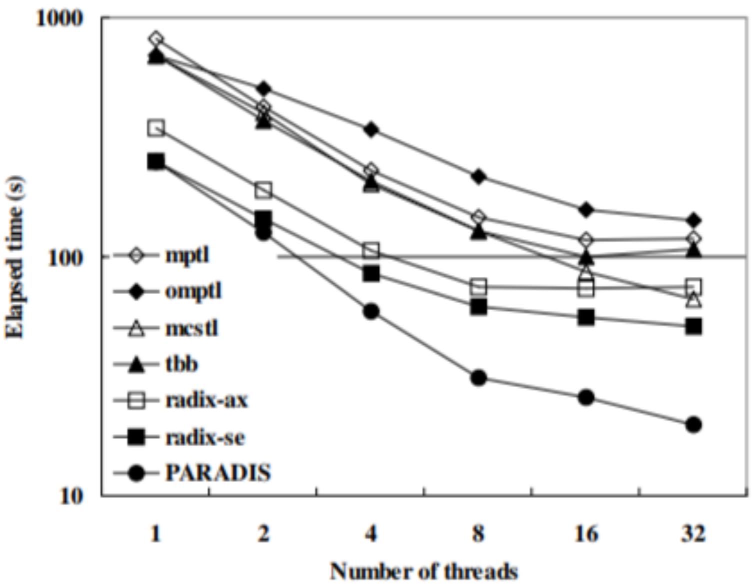
(a) the worst case for PARADIS in the 1st iteration

(b) the worst case for repairing with $r_{\{0,1\}} = w = \frac{1}{4}$

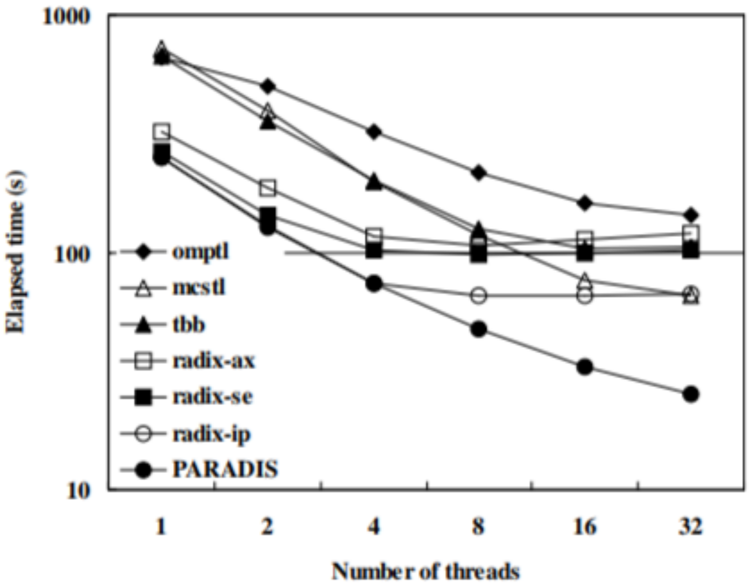
(c) the ideal case for PARADIS in the 2nd iteration

Figure 9: A pathological case for PARADIS

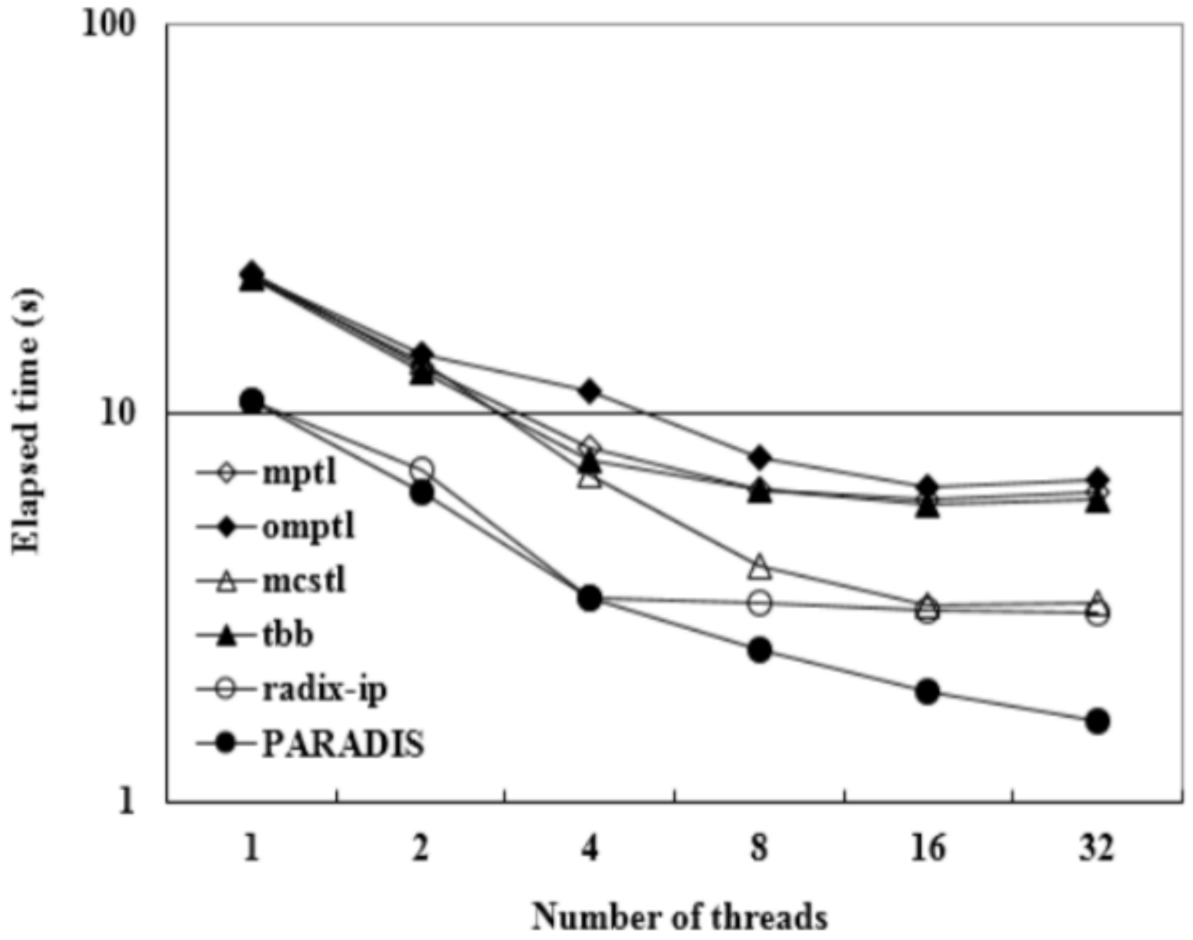
Performance



(c) Numeric random 64GB



(d) Numeric skewed (zipf 0.75) 64GB



(h) Retail sales transaction (280M records)

Final Notes

- First parallel in-place radix sort algorithm
- Eventually outperformed by a hybrid radix sort on GPUs which worked around the memory bandwidth limitation