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

1: procedure RadixSort($d[N], l$)
2: \( b = b_i \) \> Function giving bucket at level \( l \)
3: \( B = \) the range of \( b() \)
4: \( cnt[B] = 0 \) \> Histogram of bucket sizes
5: for \( n \in N \) do
6: \( cnt[b(d[n])]++ \)
7: end for
8: for \( i \in B \) do
9: \( gh_i = \sum_{j<i} cnt[j] \)
10: \( gt_i = \sum_{j\leq i} cnt[j] \)
11: end for
12: for \( i \in B \) do
13: while \( gh_i < gt_i \) do \> Till bucket \( i \) is empty
14: \( v = d[gh_i] \)
15: while \( b(v)! = i \) do
16: \( \text{swap}(v, d[gh_b(v)]++) \)
17: end while
18: \( d[gh_i++] = v \)
19: end while
20: end for
21: if \( l < L - 1 \) then \> Recurse on each bucket
22: for \( i \in B \) do
23: \( \text{RadixSort}(d[M_i], l+1) \)
24: end for
25: end if
26: 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

1: procedure PARADIS_Permute(p)
2: for $i \in B$ do
3:     $head = ph_i^p$
4:     while $head < pt_i^p$ do
5:         $v = d[head]$  \(\triangleright\) Keep moving $v$
6:         $k = b(v)$ \(\triangleright\) to its bucket $k$
7:         while $k != i$ and $ph_k^p < pt_i^p$ do
8:             swap($v$, d[$ph_k^p$]) \(\triangleright v\) into its bucket $k$
9:             $k = b(v)$ \(\triangleright\) New $v$ and $k$
10:        end while
11:     if $k == i$ then \(\triangleright\) Found a correct element
12:         d[$head++]$ = d[$ph_i^p$]
13:         d[$ph_i^p$] = $v$
14:     else
15:         d[$head++]$ = $v$
16:     end if
17: end while
18: end for
19: end procedure
Repair

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

- 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.

\[
\min: \max\{W(p) \mid \forall p\} \\
\text{where: } W(p) = \sum_{i \in B_p} \frac{C_i \cdot \log|B|C_i}{|P_i|} \\
|P_i| = |\mathcal{P}| \frac{C_i \cdot \log|B|C_i}{\sum_{j \in B} C_j \cdot \log|B|C_j}
\]

- 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

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

- $e_i C_i \leq C_i(i)$, because $e_i C_i$ represents the number of elements permuted into bucket $i$ by processors in $E_i$
Lemma 2:

\[ r_i \leq e_i \left(1 - \frac{C_i}{|\mathcal{N}|}\right), \forall i \]

- 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 \( (n \in \mathcal{M}^p_j) \).

- Therefore in bucket \( j \), any element still belonging to \( i \) must be in a stripe \( p \in E_i \).

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

\[
r_i = \frac{C_i - C_i(i)}{|\mathcal{N}|} = \frac{\sum_{j \neq i} C_j(i)}{|\mathcal{N}|} \\
\leq e_i \frac{\sum_{j \neq i} C_j}{|\mathcal{N}|} = e_i \left(1 - \frac{C_i}{|\mathcal{N}|}\right)
\]
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 \( x = y = .25 \)

\[
\begin{align*}
    r_i & \leq \min\left( \frac{C_i}{|N|} \left( 1 - e_i \right), e_i \left( 1 - \frac{C_i}{|N|} \right) \right) \\
    &= \min\left( \frac{C_i}{|N|}, e_i \right) - e_i \frac{C_i}{|N|} \leq \frac{1}{4} \\
    r &= \sum_i r_i \leq \sum_i \frac{C_i}{|N|} - \sum_i \left( \frac{C_i}{|N|} \right)^2 \\
    \text{which will be maximal with } C_i &= \frac{|N|}{|B|}, \forall i. \text{ Thus} \\
    r &\leq 1 - \frac{1}{|B|}
\end{align*}
\]
Let $w$ be the maximum fraction of wrong elements to be repaired, or
$\max\{ \sum_{i \in B_p} r_i \mid \forall p \}$

**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

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

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

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

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

$T(\mathcal{N})$ 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