Sorting on Linear Arrays

Xuan Guo
Outline

• Motivation & Models
• Sorting algorithms on linear array
  – Sorting by Comparison – Exchange
  – Sorting by Merging
• Paper
• Reference
Motivation

- Linear array is the simplest and practical parallel model;
- Algorithmic result is more valuable from weaker parallel model.

<table>
<thead>
<tr>
<th>Weak</th>
<th>Mesh</th>
<th>Hypercube</th>
<th>...</th>
<th>PRAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Linear Array</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Diagram showing weak parallel models (Linear Array, Mesh, Hypercube) and strong parallel model (PRAM)]
Model-Linear Array

• The N processors $P_1$, $P_2$, $P_3$, ..., $P_N$ are interconnected in the form of a one-dimensional array.
Definition of Sorting Problem

• Input:
  – Sequence $S = \{S_1, S_2, \ldots, S_n\}$;
  – Elements of $S$ arrive one at a time.

• Output:
  – Sorting sequence $S = \{S_1, S_2, \ldots, S_n\}$ in nondecreasing order

• Lower bound $= \Omega(n)$
Sorting by sequence algorithm

• Step 1: read $s_1$;

• Step 2: for $i$ from 2 to $n$
  – read $s_i$;
  – index of $s_i \leftarrow$ Modified Balance Search Tree ($S, s_i, k$);
  – store $s_i$ to index $k$;
  – end for
Sorting by sequence algorithm

• Analysis:
  • Run time of Modified Balance Search Tree \((S, s_i, k)\) is \(\log n\);
  • The optimal sorting sequential algorithm with time complex
    • \(O(n \log n)\)
Sorting by Comparison-Exchange

• Modified Linear Model:
  – Two-way link

• Two phases: (input/output)
  – n inputs arrive at $P_1$ one by one in first n steps
  – output n sorted elements one by one in last n steps
Sorting by Comparison-Exchange

- Algorithm:
- Step 1: $P_1$ reads $s_1$.
- Step 2: for $j=2$ to $n$ do
  - (2.1) for $i=1$ to $j-1$ do in parallel
    - $P_i$ sends its datum to $P_{i+1}$
    - End for
  - (2.2) $P_1$ reads $s_j$
  - (2.3) for all odd $i<j$ do in parallel
    - Compare-exchange($P_i, P_{i+1}$)
    - End for
  - End for
Sorting by Comparison-Exchange

- Step 3: for j=1 to n do
  - (3.1) P1 produces its datum as output
  - (3.2) for i=2 to n-j+1 do in parallel
    - Pi sends its datum to Pi-1
    - End for
  - (3.3) for all odd i<n-j do in parallel
    - Compare-exchange(Pi, Pi+1)
    - End for
  - End for
Sorting by Comparison-Exchange

• Analysis of LINEAR ARRAY COMPARISON-EXCHANGE SORT
  – $T_n = O(n)$
    • Input phase: $n$ steps
    • Output phase: $n$ steps

  – Cost=$O(n^2)$
  – Not optimal

\[
T_1^* = O(n \cdot \log n)
\]
Sorting by Merging

• Recall

• Goal
  – Finish sorting in $O(n)$ with $1+\log n$ processors $(P_1, P_2, \ldots, P_{1+\log n})$
Sorting by Merging

• Using pipelined fashion

• Assumption:
  – $n=2^r$
  – $1+r$ processors $P_1, P_2, \ldots, P_{1+r}$
Sorting by Merging

7, 2, 4, 6, 8, 3, 1, 5

P1  P2  P3  P4

P4

P3

P2

P1
Sorting by Merging

• **Algorithm - For** $P_1$
• **Step 1:** Read $s_1$ from the input sequence
• **Step 2:** $j \leftarrow 1$
• **Step 3:** for $i=2$ to $n$ do
  – (3.1) if $j$ is odd
    • Then place $s_{i-1}$ on the top output line
    • Else place $s_{i-1}$ on the bottom output line
  • End if
Sorting by Merging

– (3.2) Read \( s_i \) from the input sequence.
– (3.3) \( j<-j+1 \)
– End for

• Step 4: Place \( s_n \) on the bottom output line.

• Algorithm - For \( P_i, \ 2 <= i <= r \)
• Step 1: \( j <- 1 \)
• Step 2: \( k <-1 \)
Sorting by Merging

- **Step 3**: while \( k < n \) do
  - If the top input line contains \( 2^{i-2} \) elements
    - And the bottom input line contains one element
  - then (3.1) for \( m = 1 \) to \( 2^{i-1} \) do
    - if \( j \) is odd
      - then place the larger element on the top output line
    - else place the larger element on the bottom output line
    - end if
  - (3.2) \( j++ \)
  - (3.3) \( k \leftarrow k + 2^{i-1} \)
  - end if
- end while.

---

Where:
- \( P_1, P_2, P_3, P_4 \) represent the processing elements.
- \( 7, 2, 4, 6, 8 \) are the input elements.
- The arrows indicate the flow of elements through the processing elements.
Sorting by Merging

• Step 3: while $k < n$ do
  – If the top input line contains $2^{i-2}$ elements
    • And the bottom input line contains one element
  – then (3.1) for $m = 1$ to $2^{i-1}$ do
    • if $j$ is odd
      • then place the larger element on the top output line
      • else place the larger element on the bottom output line
    • end if
  – (3.2) $j ++$
  – (3.3) $k \leftarrow k+2^{i-1}$
  – end if
• end while.
Sorting by Merging

• Step 3: while $k < n$ do
  – If the top input line contains $2^{i-2}$ elements
    • And the bottom input line contains one element
  – then (3.1) for $m = 1$ to $2^{i-1}$ do
    • if $j$ is odd
      • then place the larger element on the top output line
    • else place the larger element on the bottom output line
    • end if
  – (3.2) $j$ ++
  – (3.3) $k \leftarrow k + 2^{i-1}$
  – end if
• end while.
Sorting by Merging

• Step 3: while $k < n$ do
  – If the top input line contains $2^{i-2}$ elements
    • And the bottom input line contains one element
  – then (3.1) for $m = 1$ to $2^{i-1}$ do
    • if $j$ is odd
      • then place the larger element on the top output line
    • else place the larger element on the bottom output line
    • end if
  – (3.2) $j$ ++
  – (3.3) $k$ <- $k+2^{i-1}$
  – end if
• end while.
Sorting by Merging

• Step 3: while \( k < n \) do
  – If the top input line contains \( 2^{i-2} \) elements
    • And the bottom input line contains one element
  – then (3.1) for \( m = 1 \) to \( 2^{i-1} \) do
    • if \( j \) is odd
      • then place the larger element on the top output line
      • else place the larger element on the bottom output line
    • end if
  – (3.2) \( j ++ \)
  – (3.3) \( k \leftarrow k + 2^{i-1} \)
  – end if
• end while.
• Step 3: while \( k < n \) do
  – If the top input line contains \( 2^{i-2} \) elements
    • And the bottom input line contains one element
  – then (3.1) for \( m = 1 \) to \( 2^{i-1} \) do
    • if \( j \) is odd
      • then place the larger element on the top output line
    • else place the larger element on the bottom output line
    • end if
  – (3.2) \( j ++ \)
  – (3.3) \( k \leftarrow k + 2^{i-1} \)
  – end if
• end while.
Sorting by Merging

- Step 3: while \( k < n \) do
  - If the top input line contains \( 2^{i-2} \) elements
    - And the bottom input line contains one element
    - then (3.1) for \( m = 1 \) to \( 2^{i-1} \) do
      - if \( j \) is odd
        - then place the larger element on the top output line
      - else place the larger element on the bottom output line
      - end if
  - (3.2) \( j \) ++
  - (3.3) \( k \leftarrow k + 2^{i-1} \)
  - end if
- end while.

\[ \begin{array}{c}
P_1 \quad P_2 \quad 4,6 \quad P_3 \quad 1,3,5,8 \quad P_4 \\
j=9 \quad j=5 \quad 2 \quad j=2 \quad 7 \end{array} \]
Sorting by Merging

• Algorithm – $P_{r+1}$
• if the top input line contains $2^{r-1}$ elements
  – And the bottom input line contains one element
• then for $m=1$ to $2^r$ do
  – Produce the larger one
  – end for
• end if
Sorting by Merging

- Algorithm – $P_{r+1}$
- if the top input line contains $2^{r-1}$ elements
  - And the bottom input line contains one element
- then for $m=1$ to $2^r$ do
  - Produce the larger one
  - end for
- end if
Sorting by Merging

• Algorithm – $P_{r+1}$
• if the top input line contains $2^{r-1}$ elements
  – And the bottom input line contains one element
• then for $m=1$ to $2^r$ do
  – Produce the larger one
  – end for
• end if
Sorting by Merging

• Algorithm – \( P_{r+1} \)
• if the top input line contains \( 2^{r-1} \) elements
  – And the bottom input line contains one element
• then for \( m=1 \) to \( 2^r \) do
  – Produce the larger one
  – end for
• end if
Sorting by Merging

• Algorithm – $P_{r+1}$
• if the top input line contains $2^{r-1}$ elements
  – And the bottom input line contains one element
• then for $m=1$ to $2^r$ do
  – Produce the larger one
  – end for
• end if
Sorting by Merging

• Algorithm – $P_{r+1}$
• if the top input line contains $2^{r-1}$ elements
  – And the bottom input line contains one element
• then for $m=1$ to $2^r$ do
  – Produce the larger one
  – end for
• end if
Sorting by Merging

• Algorithm – $P_{r+1}$
• if the top input line contains $2^{r-1}$ elements
  – And the bottom input line contains one element
• then for $m=1$ to $2^r$ do
  – Produce the larger one
  – end for
• end if
Sorting by Merging

• Algorithm – \( P_{r+1} \)

• if the top input line contains \( 2^{r-1} \) elements
  – And the bottom input line contains one element

• then for \( m=1 \) to \( 2^r \) do
  – Produce the larger one
  – end for

• end if
Sorting by Merging

• Analysis:
• Let $u$ denotes the time units elapsed;
• Processor $P_1$ produces its first output: $u=1$ ;
• For $P_i$ to start produce at $2^{i-2}+1$ after $P_{i-1}$, so it require $2^{i-1}+i-1$ time unit;
• For $P_{r+1}$ to start produce, when $u = 2^r+r$
Sorting by Merging

• Analysis:
• The last processor to terminate is $P_{r+1}$, when
• $u = (n-1) + 2^r + r$
• $= 2n + \log n - 1$

• Therefore, $t(n) = O(n)$
• $P(n) = 1 + \log n$
• Cost is $O(n \log n)$ OPTIMAL

• It obtained an $O(\log N)$-time optimal sorting algorithm using $O(N)$ processors on the LARPBS model.
LARPBS

• Linear Array with Reconfigurable Pipelined Bus System (LARPBS)
Operation on LARPBS

• One-to-one communication;
• Broadcast;
• Multiple multicast;
• Find-min-representatives operation;
• Find-max-representatives operation;
• All these can be done in $O(1)$ bus cycle
• SUP(v): 2, 3, 4, 7
• SUP(w): 1, 5, 10, 15
• UP(u): 3, 5, 7, 15
• UP(v): 2, 3, 4, 7
• UP(w): 1, 5, 10, 15
PRAM SORT

• If $\text{SUP/UP}(v) \leftrightarrow \text{SUP/UP}(w)$, then these 2 sort sequences can be merge in $O(1)$ time.
  
  – e.g. $S_a = (4, 9, 15, 18)$, local rank $(0, 1, 2, 3)$, Cross rank $(1, 2, 4, 4)$;
  
  – $S_b = (3, 8, 11, 14)$, local rank $(0, 1, 2, 3)$, Cross rank $(0, 1, 2, 2)$;

  – Local rank + Cross rank = final rank
  
  – $(3; 4; 8; 9; 11; 14; 15; 18)$
Algorithm: Pipelined Merge Sort on the LARPBS

• Begin
• for stage = 1 to 3 do
  – Step 1: Compute SUP arrays by sampling from corresponding UP arrays.
  – Step 2: Compute SUP(v) → UP(u) and SUP(w) → UP(u).
  – Step 3: Compute SUP(v) ↔ SUP(w).
  – Step 4: Compute NEWUP(u) = SUP(v) ∪ SUP(w).
  – Step 5: Maintain ranks NEWUP(u) → NEWSUP(v) and NEWUP(u) → NEWSUP(w).
• endfor
• End
Step 2: Compute SUP(v) → UP(u) and SUP(w) → UP(u).

- Do find-min-representatives operation in UP(u)
- UP(u) sends its item and rank $s$ to $r^{th}$ processors in SUP(v)
- Example:
  - UP(u): 3, 5, 7, 15; SUP(v): 2, 3, 4, 7
  - rank $s$ in UP(u): 0, 1, 2, 3
  - rank $r$ in SUP(v): 1, 3, 3, 4
  - representative: 3, 5, 15
Step 3: Compute \( \text{SUP}(v) \leftrightarrow \text{SUP}(w) \).

- Do find-max-representatives operation in \( \text{SUP}(v) \) based on item’s ranks in \( \text{UP}(u) \);
- Get straddle \( P_j \) and \( P_{j+1} \) in \( \text{UP}(u) \) for \( P_i \);
- Send ranks of \( P_j \) and \( P_{j+1} \) to \( \text{SUP}(w) \) to get items back to \( P_i \)
- Do comparison
Analysis

• Step 1: simple local comparison + a one-to-one communication;
• Step 2: find-min-representative + one-to-one communication + broadcast;
• Step 3: find-max-representative + multiple multicast + one-to-one communication + broadcast;
• Step 4: local addition + a one-to-one communication;
• Step 5: constant number of one-to-one communication and similar communications and computations as Step 3;
• There is an \( O(\log N) \) optimal sorting algorithm on the LARPBS model using \( O(N) \) processors.
State of art

• The systems edge of the Parameterized Linear Array with a Reconfigurable Pipelined Bus System (LARPBS(p)) optical bus parallel computing model. The Journal of Supercomputing Volume 48, Number 2, 183-209, 2009;
  – The main part of this paper considers some practical systems related aspects of the LARPBS(p) model, specifically: fiber and free space based implementations, feasibility study, communication traffic analysis, bus collision avoidance, and the cost analysis of a MIMD algorithm. An overview perspective of the work is presented, thereby: the edge of systems related research is identified.

• Min He; Xiaolong Wu; Si Qing Zheng; Englert, B.; , "Optimal Sorting Algorithms for a Simplified 2D Array with Reconfigurable Pipelined Bus System," Parallel and Distributed Systems, IEEE Transactions on , vol.21, no.3, pp.303-312, March 2010;
  – To increase the scalability of the LARPBS model, it proposed a two-dimensional extension: a simplified two-dimensional Array with Reconfigurable Pipelined Bus System (2D ARPBS). While achieving better scalability, it showed the effectiveness of this newly proposed model by designing two novel optimal sorting algorithms on this model.
• Thank You
• Q & A
Reference

• The systems edge of the Parameterized Linear Array with a Reconfigurable Pipelined Bus System (LARPBS(p)) optical bus parallel computing model. The Journal of Supercomputing Volume 48, Number 2, 183-209, 2009;
• Min He; Xiaolong Wu; Si Qing Zheng; Englert, B.; , "Optimal Sorting Algorithms for a Simplified 2D Array with Reconfigurable Pipelined Bus System," Parallel and Distributed Systems, IEEE Transactions on , vol.21, no.3, pp.303-312, March 2010;