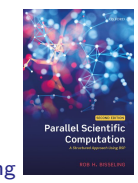
Sorting

Section 1.8 of Parallel Scientific Computation, 2nd edition

Rob H. Bisseling

Utrecht University



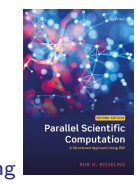
Recursive function Quicksort

input: $\mathbf{x}$: vector of length n, interval $[lo, hi]$, $0 \leq lo \leq hi < n$.
output: $\mathbf{x}$ is sorted with $x_i \leq x_j$ for all i, j with $lo \leq i \leq j \leq hi$.

function QUICKSORT($\mathbf{x}, lo, hi$)

$i := \text{Split}(\mathbf{x}, lo, hi)$;
if $i - 1 > lo$ **then**
    Quicksort($\mathbf{x}, lo, i - 1$);
if $i + 1 < hi$ **then**
    Quicksort($\mathbf{x}, i + 1, hi$);

C. A. R. Hoare, *Communications of the ACM*, **4**(7) (1961), p. 321.



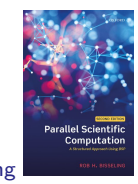
Splitter

- Index r, with $0 \leq r < n$, is a splitter if

$$x_i \leq x_r \text{ for } i < r,$$
$$x_i \geq x_r \text{ for } i > r.$$

- The vector **x** of length $n = 10$ has one splitter, $i = 5$, with splitting value $x_5 = 8$:

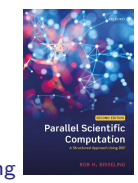
$x_i =$	3	6	2	7	5	8	13	14	10	11
$i =$	0	1	2	3	4	5	6	7	8	9



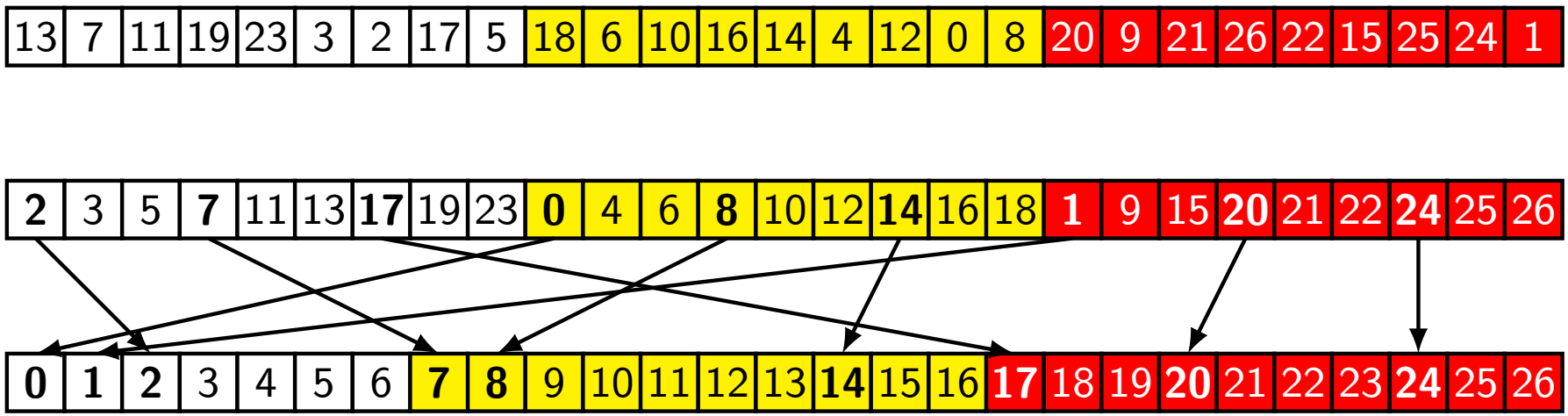
Splitting a vector based on a random pivot

function SPLIT($\mathbf{x}, lo, hi$)

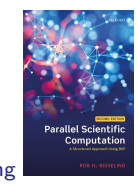
pick piv, with $lo \leq piv \leq hi$;
$val := x_{piv}$;
swap(x_{piv}, x_{hi});
$i := lo$;
for $j := lo$ **to** $hi - 1$ **do**
  { Loop invariant:
    $x[lo, i) < val$
    $x[i, j) \geq val$
    $x[j, hi)$ not yet processed }
  if $x_j < val$ **then**
    swap(x_i, x_j);
    $i := i + 1$;
swap(x_i, x_{hi});
return i;



Parallel regular sample sort



H. Shi and J. Schaeffer, *Journal of Parallel and Distributed Computing*, **14**(4) (1992), pp. 361–372.



BSP sorting algorithm: supersteps 0, 1

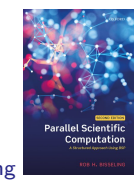
input: **x** : vector of length n with $n \bmod p^2 = 0$,
$x_i \neq x_j$ for all $i \neq j$ (no ties).
x is block distributed with block size $b = n/p$.

output: **x** is sorted with $x_i \leq x_j$ for all $i < j$.
x is block distributed with variable block size $b_s \leq 2b$.

{ Sort the local block and create samples } ▷ Superstep (0)
Quicksort$(x, sb, (s+1)b-1)$;
for $i := 0$ **to** $p-1$ **do**
    $sample_s[i] := x[sb + i \cdot \frac{n}{p^2}]$;

{ Broadcast the samples } ▷ Superstep (1)
for $t := 0$ **to** $p-1$ **do**
    put $sample_s$ in $P(t)$;

...



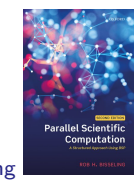
Cost analysis: supersteps 0, 1

- Assumption: $n \bmod p = 0$, so each processor has a block of exactly $\frac{n}{p}$ array elements.
- Assumption: $\frac{n}{p} \bmod p = 0$, so each processor has p subblocks of exactly $\frac{n}{p^2}$ array elements.
- Sorting an array of length $\frac{n}{p}$ costs

$$T_{(0)} = \frac{n}{p} \log_2 \frac{n}{p} + l.$$

- Broadcasting p local samples to $p-1$ other processors costs

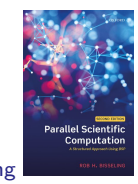
$$T_{(1)} = p(p-1)g + l.$$



BSP sorting algorithm: superstep 2

{ Concatenate and sort the samples }
for $t := 0$ **to** $p - 1$ **do**
    for $i := 0$ **to** $p - 1$ **do**
        $sample[tp + i] := sample_t[i]$;
    $start[t] := tp$;
$start[p] := p^2$;
Mergesort($sample, start, p$);

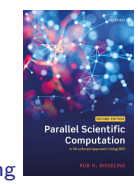
{ Create splitters }
for $t := 0$ **to** $p - 1$ **do**
    $splitval[t] := sample[tp]$;
$splitval[p] := \infty$;



Cost analysis: superstep 2

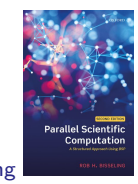
- The p^2 samples are already arranged as p sorted parts, so we use a mergesort instead of a quicksort.
- Mergesort repeatedly merges a pair of sorted parts, in $\lceil \log_2 p \rceil$ phases, each costing p^2 flops.
- Mergesort($\mathbf{x}, start, p$) sorts the vector $\mathbf{x}$ using already sorted intervals $[start[t], start[t+1]-1]$, for $t = 0, \ldots, p-1$.
- The total cost of the mergesort of the samples is

$$T_{(2)} = p^2 \lceil \log_2 p \rceil + l.$$



BSP sorting algorithm: superstep 3

{ Split the local block and send the resulting parts }
for $t := 0$ **to** $p - 1$ **do**
    { Contribution from $P(s)$ to $P(t)$ }
    $X_{st} := \{x_i \ : \ sb \leq i < (s+1)b \wedge$
        $splitval[t] \leq x_i < splitval[t+1]\}$;
    put X_{st} in $P(t)$;

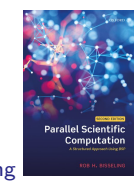


Cost analysis: superstep 3

- Each processor sends at most all its data (b values), and receives at most b_s data, so

$$T_{(3)} = \max_s \max(b, b_s)g + l.$$

- b_s is the block size of $P(s)$ on output.

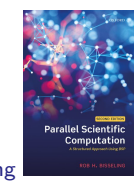


BSP sorting algorithm: superstep 4

{ Concatenate the received parts }
$X_s := \cup_{t=0}^{p-1} X_{ts}$;

{ Sort the local block }
$start_s[0] := 0$;
for $t := 1$ **to** p **do**
    $start_s[t] := start_s[t-1] + |X_{t-1,s}|$;
$b_s := start_s[p]$;

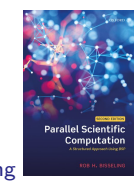
Mergesort$(X_s, start_s, p)$;



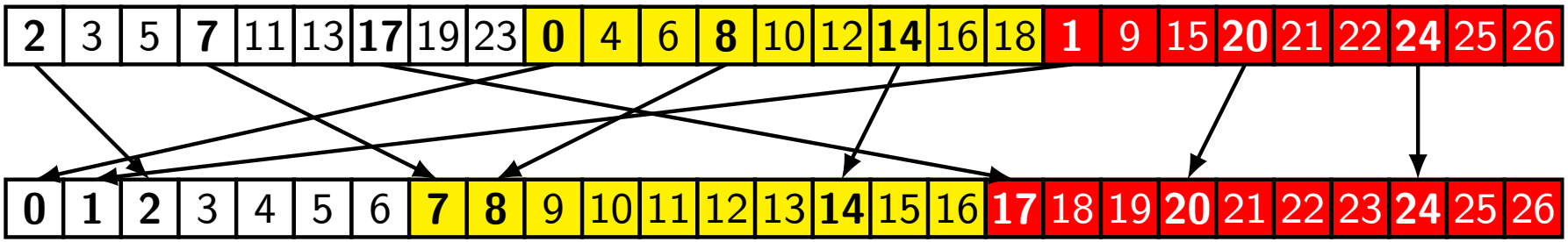
Cost analysis: superstep 4

- The starts are computed in p operations.
- Mergesort repeatedly merges a pair of sorted parts, in $\lceil \log_2 p \rceil$ phases, each accessing at most all the b_s local data, so

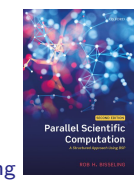
$$T_{(4)} = p + \max_s b_s \lceil \log_2 p \rceil + l.$$



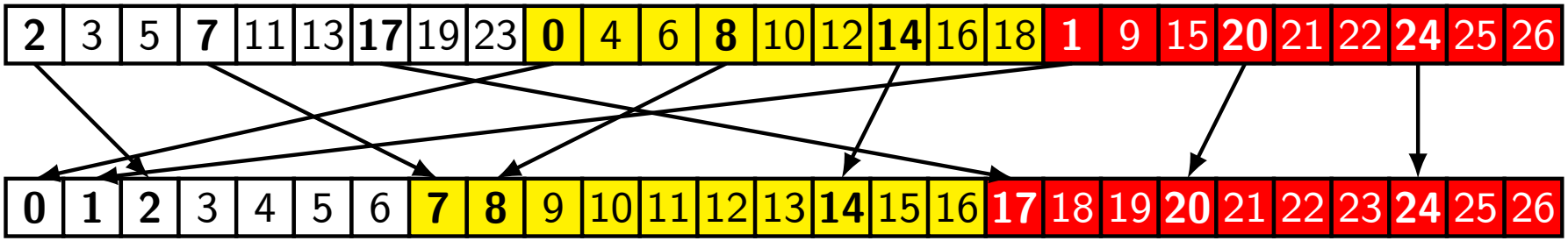
Proof that $b_s \leq 2b$



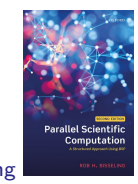
- A subblock is a part of the locally sorted vector of length $\frac{n}{p^2}$ that starts with a sample.
- There are p local subblocks.
- Subblock (**17**,19,23) with sample **17** contributes to $P(2)$.
- Consider a fixed processor $P(s)$, with b_s output data.
- The local output block contains exactly p samples, and hence exactly p subblocks contribute a sample. In total, these subblocks contribute at most $p \cdot \frac{n}{p^2} = \frac{n}{p} = b$ data values.



Proof that $b_s \leq 2b$ (cont'd)



- Thinking alert! Now consider contributions by a subblock that does not contribute its sample.
- Example: subblock (**14**,16,18) only contributes 18 to $P(2)$ but not its sample **14**.
- Each processor $P(t)$ can contribute at most one such subblock to $P(s)$, because the subblock must have a value $< splitval[s]$ and a value $\geq splitval[s]$. (Other subblocks of processor $P(t)$ are completely to the left or right.)
- In total, these subblocks contribute at most $p \cdot \frac{n}{p^2} = \frac{n}{p} = b$ data values.
- By adding two terms b, we obtain the bound $b_s \leq 2b$.

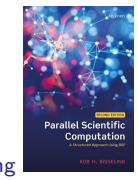


Total BSP cost

$$
\begin{aligned}
T_{\text{samplesort}} &= \frac{n}{p}\log_2\frac{n}{p} + p^2\lceil\log_2 p\rceil + \frac{2n}{p}\cdot\lceil\log_2 p\rceil + p \\
&\quad + \left(p(p-1) + 2\frac{n}{p}\right)g + 5l \\
&\approx \frac{n\log_2 n}{p} + p^2\log_2 p + \left(p^2 + 2\frac{n}{p}\right)g + 5l.
\end{aligned}
$$

- We approximate and drop lower-order terms in the first cost expression, to obtain a more insightful expression.
- We follow Richard Hamming's motto:

  The purpose of computing is insight, not numbers.



Total BSP cost

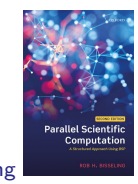
$$T_{\text{samplesort}} \approx \frac{n \log_2 n}{p} + p^2 \log_2 p + \left(p^2 + 2\frac{n}{p}\right) g + 5l.$$

- If $p \leq n^{1/3}$, then $p^2 \leq \frac{n}{p}$ and hence

$$p^2 \log_2 p \leq \frac{n}{p} \log_2 n,$$

  so that the computation is efficient.
- Otherwise, sorting the samples would dominate.
- If the condition holds, the communication cost is dominated by the term $2\frac{n}{p}g$.
- For communication to be efficient, this term should not exceed the main computation term, i.e.,

$$2g \leq \log_2 n.$$



Summary

- Parallel samplesort uses samples at regular intervals to split local data into p subblocks.
- These subblocks are used to send the data to their destination processor.
- The output block sizes are imbalanced, but at most by a factor of 2.
- Further oversampling can reduce this factor.
- We presented a BSP samplesort algorithm with 3 computation supersteps and 2 communication supersteps.