COMP 790-033 - Parallel Computing

Lecture 11
Oct 26, 2022

BSP (2)
Parallel Sorting in the BSP model

Topics

1. What work remains this semester:
   - programming project and presentation

2. Sorting in the BSP model

Parallel sorting: problem definition

- **Given**
  - N values, each of size b bits
  - a total order $\leq$ defined on the values
- **Initial distribution**
  - each processor holds $n = N / p$ values
- **Result**

$proc_0$	$proc_1$	$proc_2$	...	$proc_{p-1}$
V_1	V_{k_1+1}	V_{k_2+1}		$V_{k_{p-1}+1}$
...	...	...		...
V_{k_1}	V_{k_2}	V_{k_3}		V_{k_p}

  - $V_i \leq V_{i+1}$ for all $1 \leq i < N = k_p$
  - generally $k_i = n \cdot i$, i.e. evenly distributed across processors

Parallel sorting: general remarks

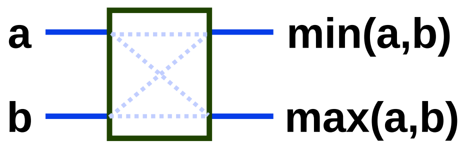
- **Typically concerned with case of *N* >> *p***
  - Small *N* problems don't require parallel processing
  - Use algorithm cascading with efficient sequential sort of *n* elements
    - » sequential radix sort of *n* values has $W^{\text{SORT}}(n) = \Omega(bn)$
    - » sequential comparison-based sort has $W^{\text{SORT}}(n) = \Omega(n \lg n)$ and may be more appropriate when *b* is large
  - Examine scalability in *N* and *p* using BSP model
    - » two parallel algorithms considered
      - Bitonic sort, Sample sort

- **What is the lower bound BSP cost for sorting?**
  - Work bound
    - » (1/p) * optimal sequential work $W^{\text{SORT}}(N)$
  - Communication bound
    - » each value may have to move between processors from input to output
  - BSP lower bound

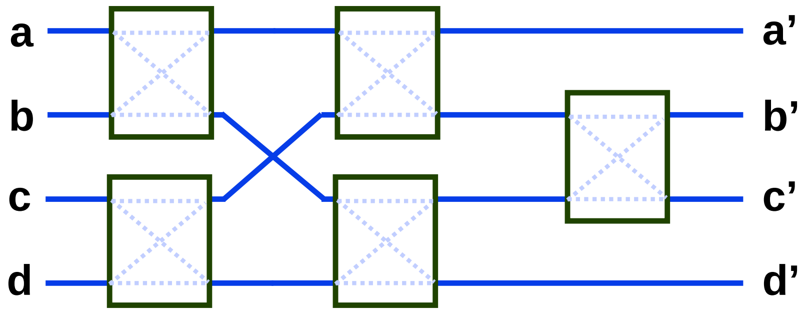
$$C_p^{\text{SORT}}(N,p) \geq \frac{W^{\text{SORT}}(N)}{p} + \frac{N}{p} \cdot g + L$$

Background: Sorting networks for parallel sorting

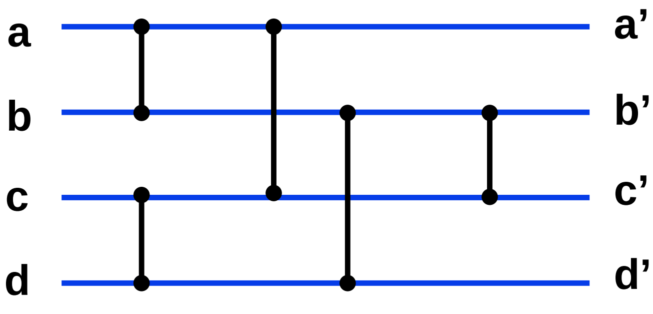
- **Basic component: the *comparator* module**



- **Comparator modules can be connected to form a sorting network**
  - all inputs are presented in parallel
    - » ex: sorting network for 4 values



sorting network

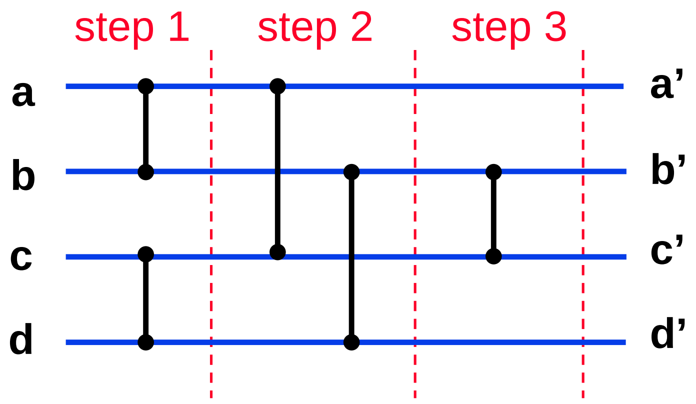


schematic representation

Sorting networks

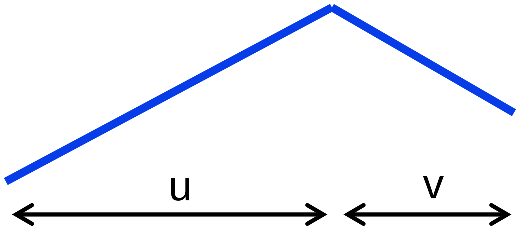
- **Sorting networks are *oblivious***
  - predetermined sequence of comparisons sorts *any* input sequence
  - the depth of a comparator is the maximum number of preceding comparators on any path to an input

- **A sorting network specifies a parallel sorting algorithm**
  - in step i, evaluate all comparators at depth i in parallel
    - » each step permutes inputs to outputs (EREW)
    - » at most n comparators evaluated in each step
      - let d(n) be the depth of a network of size n, then S(n) = d(n), W(n) = O(n·d(n))

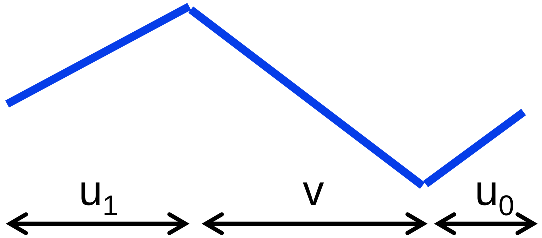


Bitonic Sequence

- **Definitions**
  - A sequence of values *w* is up-down if *w* = *uv* with *u* increasing and *v* decreasing
    - » ex: w = 1 3 5 9 6 4 3



  - A sequence of values *w* is bitonic if *w* is a circular rotation of an up-down sequence
    - » ex: w = 5 9 6 4 3 1 3



Bitonic sequence theorem

- **Theorem**
  - Suppose *w* is a bitonic sequence of length 2*n* and we define sequences r, s of length *n* as follows

$$r_i = \min(w_i, w_{n+i})$$
$$s_i = \max(w_i, w_{n+i})$$

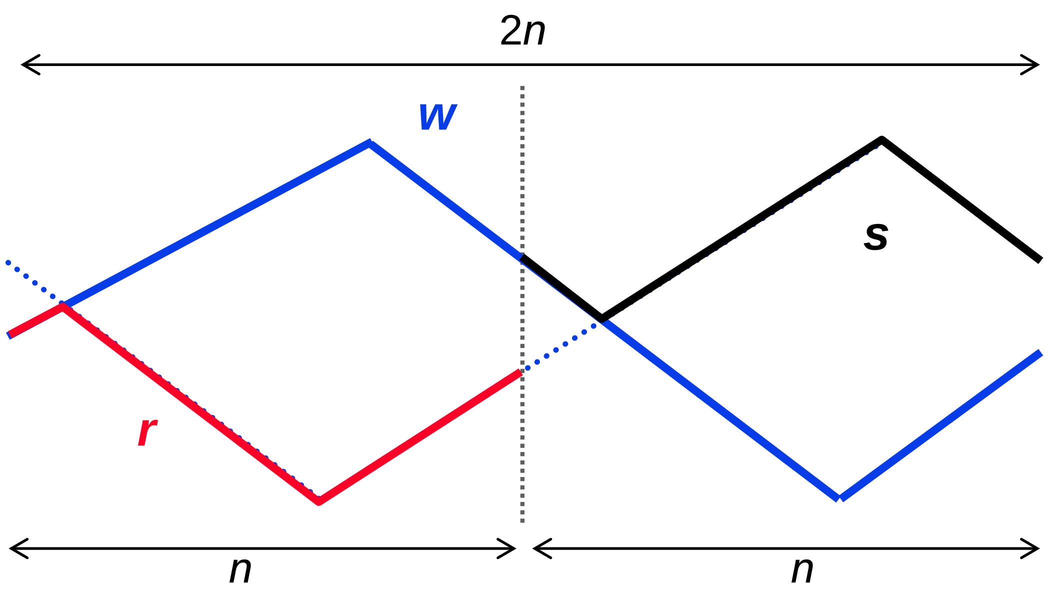
then

(1) $\forall 1 \le i, j \le n: \quad r_i \le s_j$ ← partitions the sorting problem !

(2) r, s are both bitonic sequences ← bitonic subproblems !

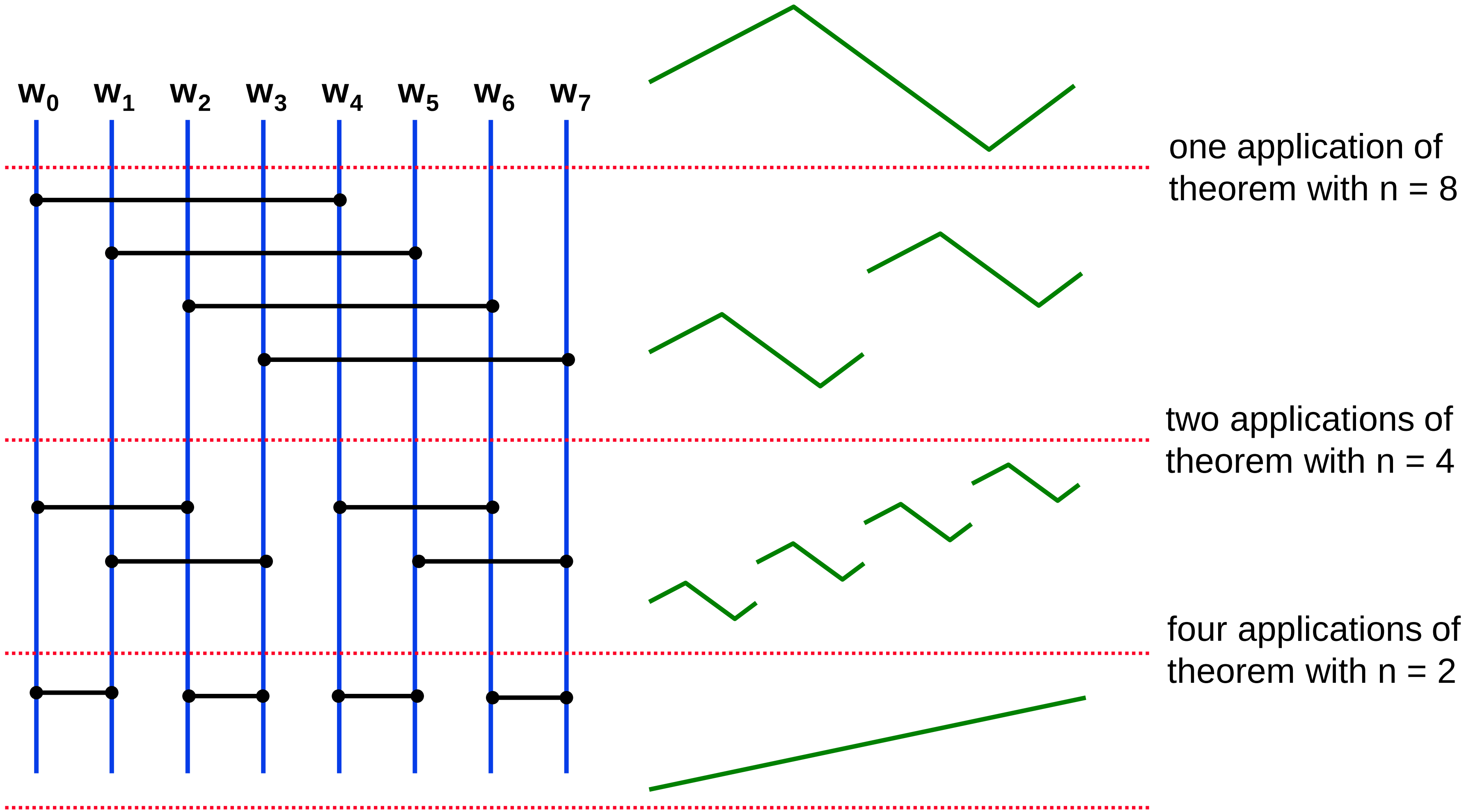
- **Proof**

  (by picture)



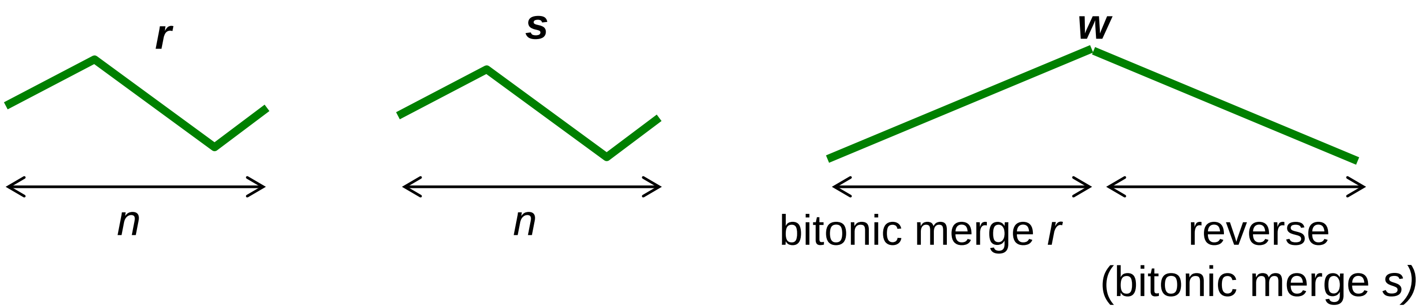
Bitonic merge

- A bitonic sequence of length $n = 2^k$ can be sorted with a depth k sorting network
  - apply bitonic sequence theorem recursively

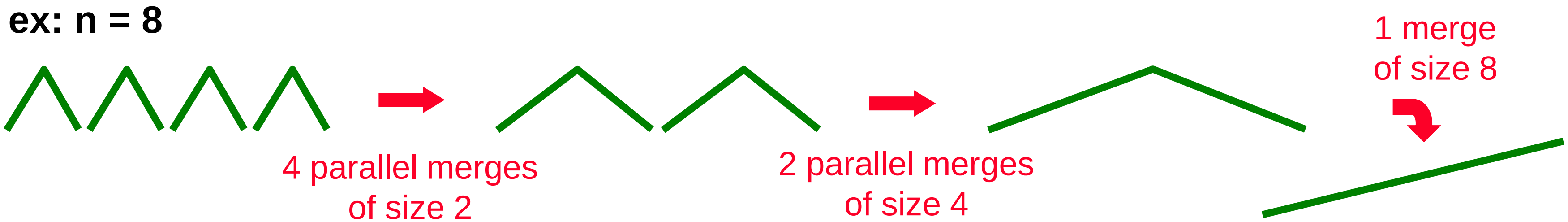


Bitonic Sort

- **Combine two length n bitonic merge sequences to form a length 2n bitonic sequence**
  - given two bitonic sequences *s, r* of length *n* let
    w = (bitonic merge *r*) ++ (reverse (bitonic merge *s*))
  - w is a bitonic sequence of length 2n



- **Bitonic sort of n = 2^k values**
  - view input as n/2 bitonic sequences of length 2
  - combine bitonic sequences k-1 times to create a length n bitonic sequence
  - apply final bitonic merge to yield sorted sequence
- **ex: n = 8**

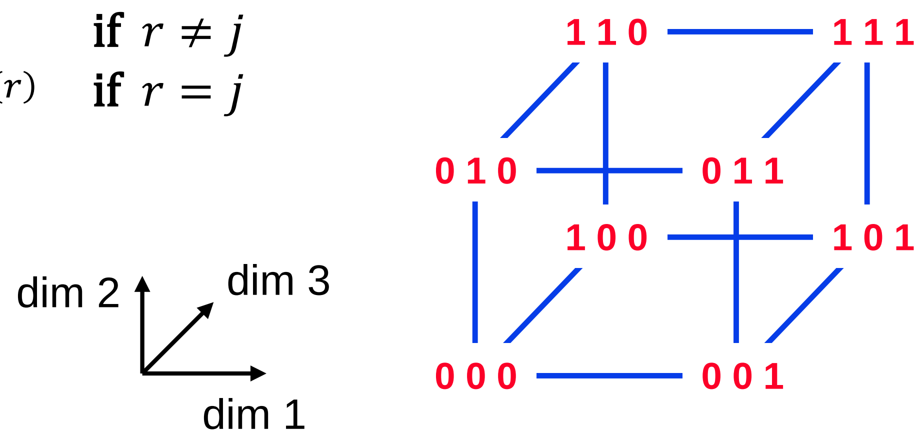


Hypercube communication pattern

- Let $p = 2^k$ for some $k \geq 0$. Processors are numbered $0 \leq h < p$. Let $h^{(j)}$ be the j^{th} bit in the boolean representation of h, where $1 \leq j \leq k$
  - ex $p = 8, k = 3$ $h^{(3)}$ $h^{(1)}$
    $h = 4 =$ 1 0 0

- For $0 \leq h < p$, processor $nb_j(h)$ is the neighbor of processor h in dimension j. The bits of $nb_j(h)$ are specified as follows, for $1 \leq r \leq k$

$$\{nb_j(h)\}^{(r)} = \begin{cases} h^{(r)} & \textbf{if } r \neq j \\ 1 - h^{(r)} & \textbf{if } r = j \end{cases}$$



Bitonic sort of *A*[0:*p*-1] using *p* processors

- **Assumptions**
  - $p = 2^k$ and *A*[*h*] is stored in variable *a* on processor *h*
  - CE(x,y) = (min(x,y), max(x,y))

- **SPMD program for processor h**

```
for i := 1 to k do
    for j := i downto 1 do
        b := value of a at nb_j(h)
        -----------------------------------
        a,b := CE(a,b)
        if (h^(j) ≠ h^(i+1)) then a,b := b,a
        -----------------------------------
    end do
end do
```

2 supersteps

- **BSP cost** $C(p) = \sum_{i=1,k} \sum_{j=1,i} \left(O(1) + 1 \cdot g + 2 \cdot L\right)$

$$= \left(O(1) + 1 \cdot g + 2 \cdot L\right) \sum_{i=1,k} \sum_{j=1,i} 1 \quad = \quad \left(O(1) + 1 \cdot g + 2 \cdot L\right) \frac{k(k+1)}{2}$$

$$= O(\lg^2 p)\left(1 + g + L\right)$$

Extending bitonic sort to $N > p$

- **Simulate larger parallel machine**
  - Let $N = np$ where $n = 2^q$ and $p = 2^k$ so $N = 2^{(k+q)}$

```
for i:= 1 to k+q do
    for j := i downto 1 do
        CE on dimension j
```

- **BSP cost of CE on dimension j**
  - lower dimensions in memory, higher dimensions across processors

$$T_j(n) = \begin{cases} O(n), & \text{if } j \le q \\ O(n) + n \cdot g + L, & \text{if } j > q \end{cases}$$

- **BSP cost for algorithm** $C(N,p) = \sum_{i=1}^{k+q} \sum_{j=1}^{i} T_j(N/p)$

$$= \left(\frac{(\lg N)(1+\lg N)}{2} \right) \cdot O\left(\frac{N}{p}\right) + \sum_{i=q+1}^{k+q} \sum_{j=q+1}^{i} \left(\frac{N}{p} \cdot g + 2L \right)$$

$$= \Theta(\lg^2 N) \cdot \frac{N}{p} + \Theta(\lg^2 p) \cdot \left(\frac{N}{p} \cdot g + 2L \right)$$

Improving work-efficiency

- **What can be done?**
  - first q iterations of outer loop create sorted sequences in processor memories
    - » replace with efficient localsort (O(n) radix sort is assumed here for simplicity)
  - for each value i > q in outer loop, last q iterations of inner loop perform a bitonic merge in processor memories
    - » replace with efficient O(n) sequential algorithm for bitonic merge (`sbmerge`)

- **Updated program**

```
localsort(n)
for i:= q+1 to k+q do
      for j := i downto q+1 do
         CE on dimension j
      sbmerge(n)
```

- **BSP cost**

$$C(N,p) = \Theta\left(\frac{N}{p}\right) + (\lg p)\left(\frac{1+\lg p}{2}\cdot\left(O\left(\frac{N}{p}\right)+\frac{N}{p}\cdot g + 2L\right) + O\left(\frac{N}{p}\right)\right)$$

$$= \Theta(\lg^2 p)\cdot\frac{N}{p} + \Theta(\lg^2 p)\cdot\left(\frac{N}{p}\cdot g + L\right)$$

Improving communication efficiency

- **What can be done?**
  – combine communication for up to lg p successive CE operations

- **Updated program**

```
localsort(n)
for i:= q+1 to k+q do
      transpose(n)
      (i-q) successive CE(n) on local data
      transpose(n)
      sbmerge(n)
```

- **BSP cost**

$$C(N,p) = \Theta\left(\frac{N}{p}\right) + (\lg p)\left(2\left(\frac{N}{p}\cdot g + L\right) + (1+\lg p)\cdot\Theta\left(\frac{N}{p}\right) + \Theta\left(\frac{N}{p}\right)\right)$$

$$= \Theta(\lg^2 p)\cdot\frac{N}{p} + \Theta(\lg p)\cdot\left(\frac{N}{p}\cdot g + L\right)$$

BSP predicted and measured times for bitonic sort

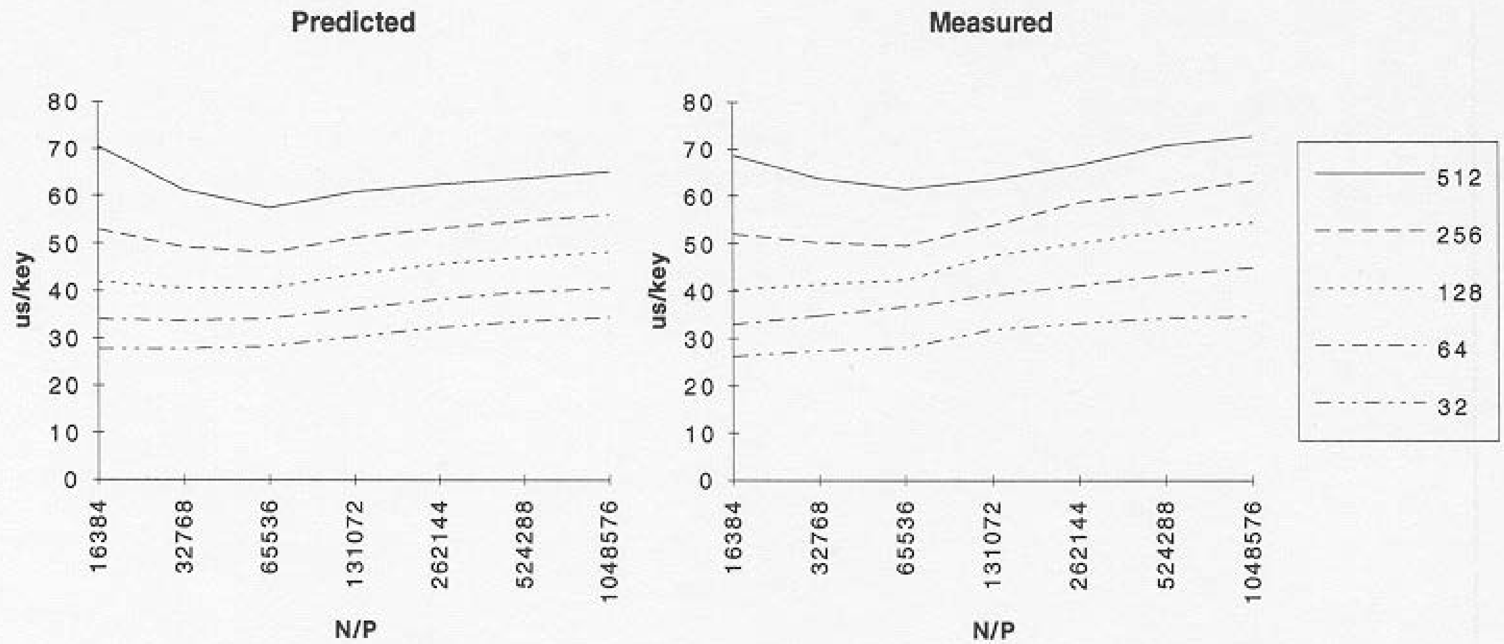


Figure 1.4 *Predicted and measured execution time per key of bitonic sort on the CM-5. Times are shown to sort between 16K and 1M keys per processor on 32, 64, 128, 256 and 512 processors.*

BSP breakdown of time in optimized bitonic sort

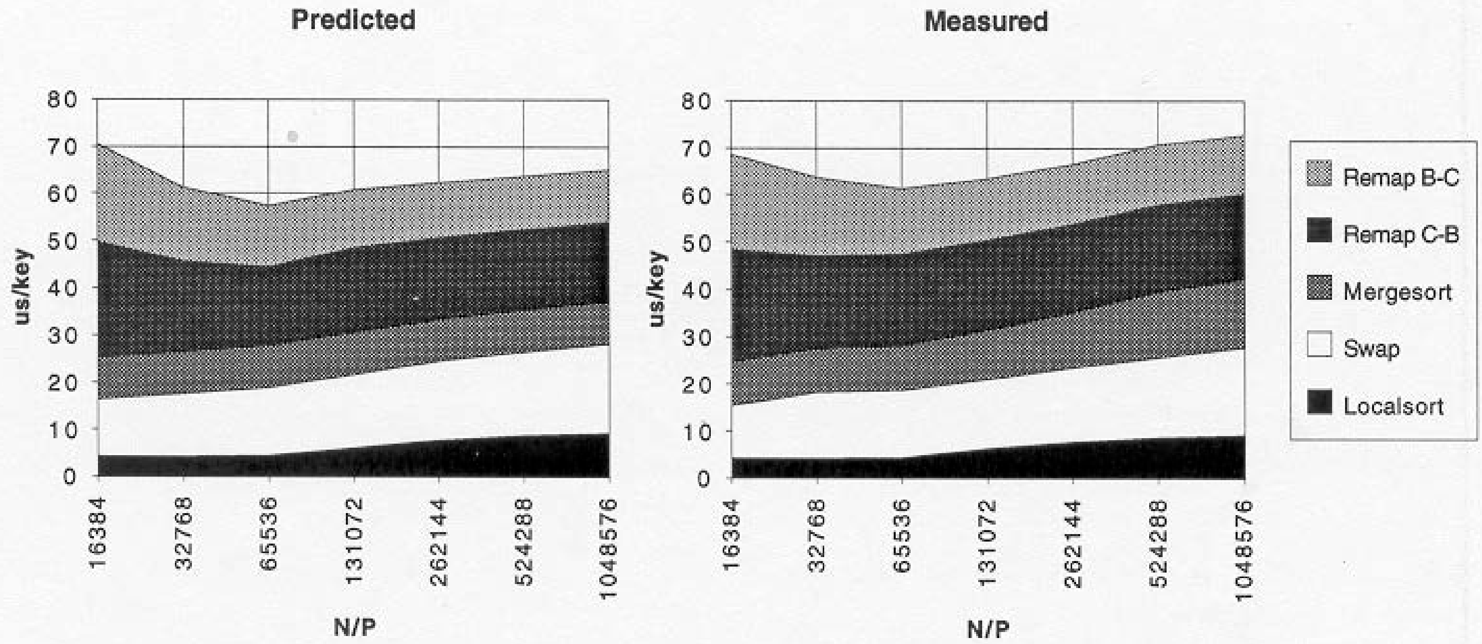


Figure 1.5 *Predicted and measured execution times per key on 512 processors for the phases of bitonic sort. The time for the single gather is included in the time for the remap from a blocked to a cyclic layout; likewise, the time for the scatter is included in the time for the remap from a cyclic to a blocked layout.*

Probabilistic parallel sorting algorithms

- **Definitions**
  - An unordered collection H with N disjoint values is partitioned by splitters $S = S_1 < ... < S_{p\text{-}1}$ into p disjoint subsets $H_1 \ldots H_p$ such that

$$H_i = \{h \mid h \in H \text{ and } S_{i-1} \le h < S_i\} \qquad (\text{define } S_0 = -\infty, \text{ and } S_p = +\infty)$$

  - The skew $W(S)$ of a partition S is the ratio of the maximum partition size to the optimal partition size (N/p)

$$W(S) = \max_{1 \le i \le p} \left(\frac{|H_i|}{N/p} \right)$$

Determining good splitters through sampling

- **Determining a set of splitters through sampling**
  - sample $k \cdot p$ elements at random from H
    - » $k \geq 1$ is the oversampling ratio
  - sort this sample into order $b_1 < b_2 < \ldots < b_{k \cdot p}$ and choose $S_i = b_{k \cdot i}$

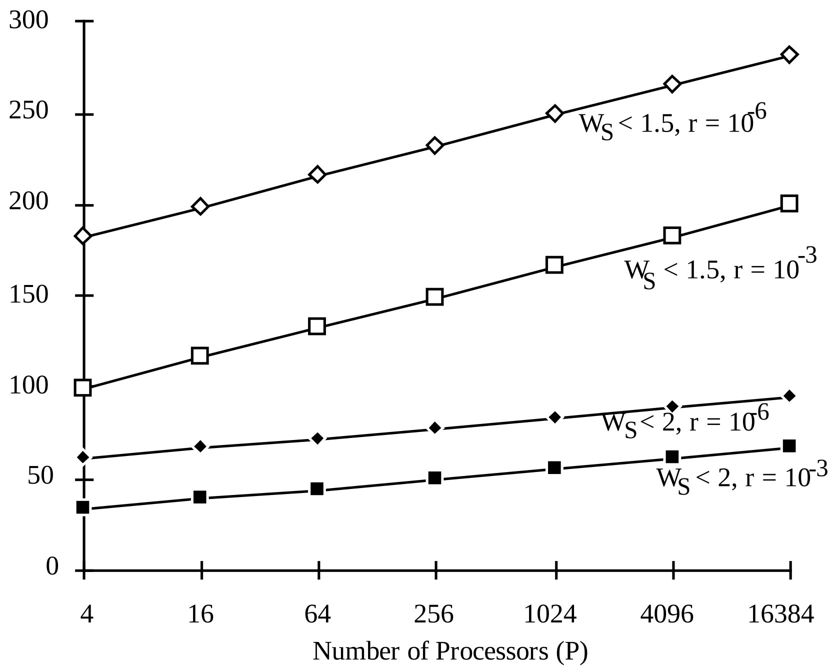
- **Probabilistic bounds on W(S) of a sampled set of splitters S**
  - given some maximum skew W and a failure probability 0 < r < 1

$$\Pr\left(W(S) > W\right) \leq r \quad \text{when} \quad k \geq \frac{2\ln(p/r)}{\left(1-1/W\right)^2 W} \quad (\text{provided } p > 1, \quad W > 1.3)$$

  - if we oversample sufficiently in choosing a set of splitters, the chance of a large skew can be made arbitrarily small

Oversampling ratio *k* as a function of *p*

- **Example**
  - for p = 100 processors, we need to sample k = 4 ln (p/r) = 74 values per processor to bound the skew W(S) < 2 with failure probability $r = 10^{-6}$



Parallel samplesort

- **Algorithm**
  1. sample k values at random in each processor to limit skew W w.h.p.

     O(k)
  2. sort kp sampled keys, extract p-1 splitters, and broadcast to all processors
     a) by sending all samples to one processor and performing a local sort

        O(kp) + (k+2)p · g + 2 · L

     a) by performing a bitonic sort with k values per processor

        O(k $\lg^2$ p) + k(1+2 lg p) · g + (1+lg p) · L
  3. compute destination processor for each value by binary search in splitter set

     O(N/p lg p)
  4. permute values

     WN/p · g + L
  5. perform local sort of values in each processor

     O(Ts(WN/p))

- **BSP cost**

$$C^{\text{SAMPLE}}(N,p,W) = \Theta\left(W + \lg p\right)\left(\frac{N}{p}\right) \quad + \quad W\left(\frac{N}{p}\right)\cdot g \quad + \quad (\lg p)\cdot L$$
$$+ \quad O\left(k \lg p\right)\left(\lg p \cdot g + L\right)$$

Samplesort: predicted and measured times

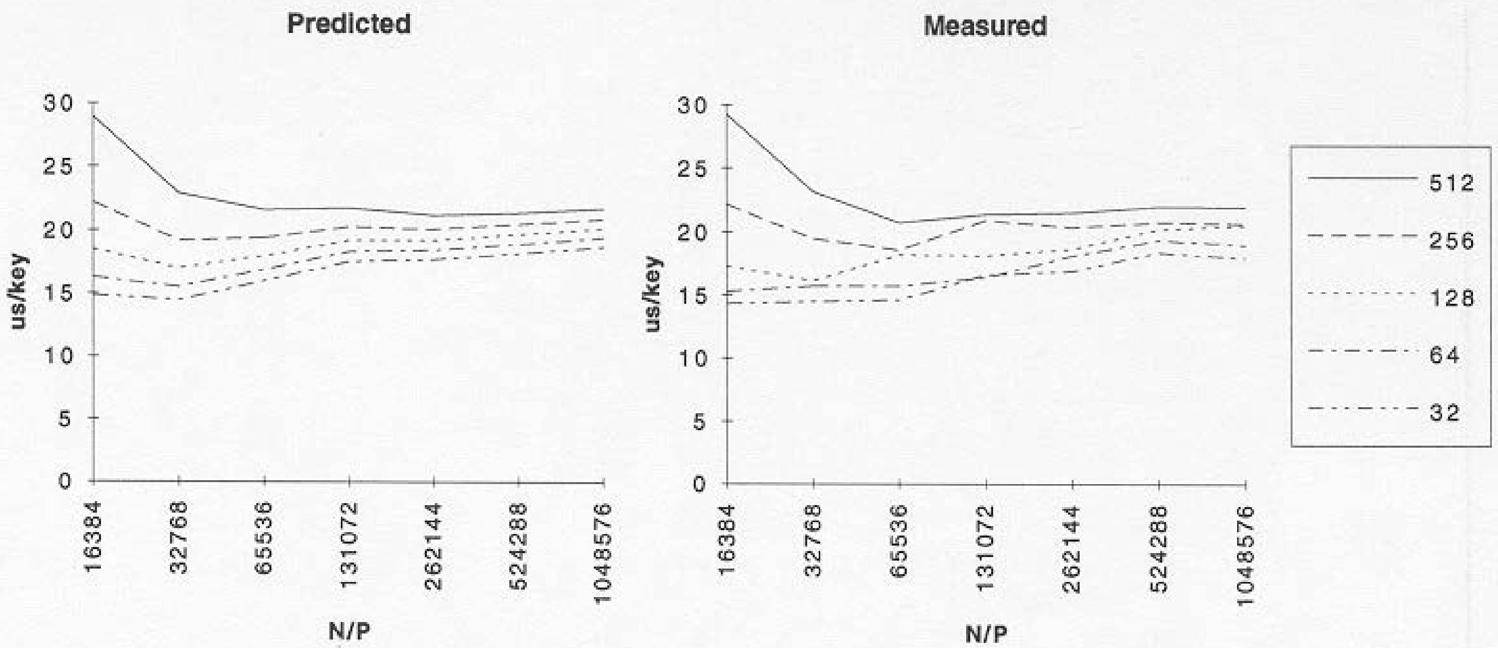


Figure 1.12 *Estimated and measured execution time of parallel sample sort on the CM-5.*

Samplesort: breakdown of execution time

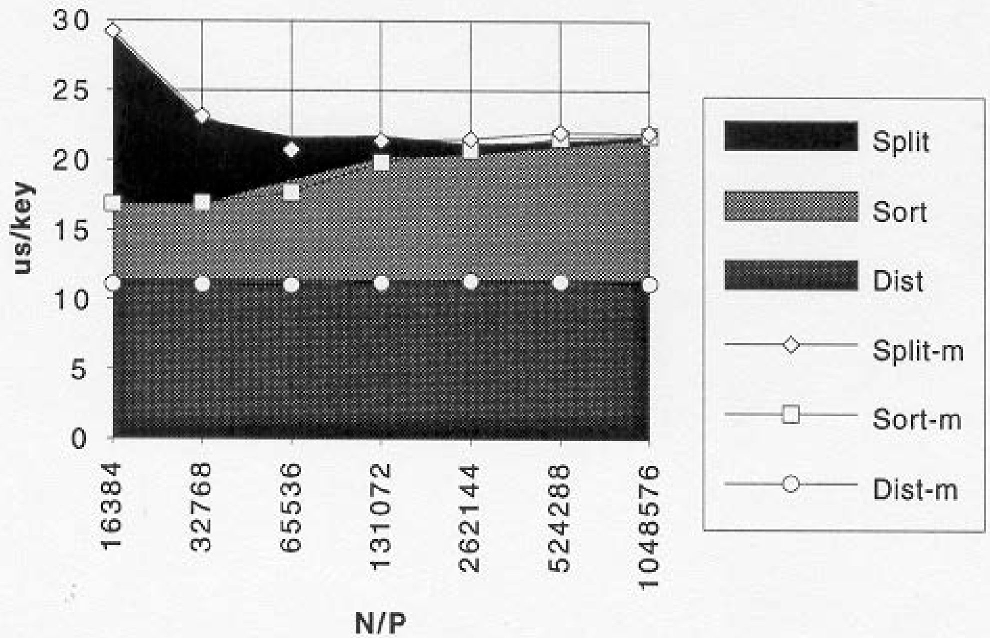


Figure 1.13 *Estimated and measured execution times of various phase of parallel sample sort on 512 processors.*

Parallel sorting: performance summary

- **32 bit values**
  – for small N/p (not shown), bitonic sort is superior

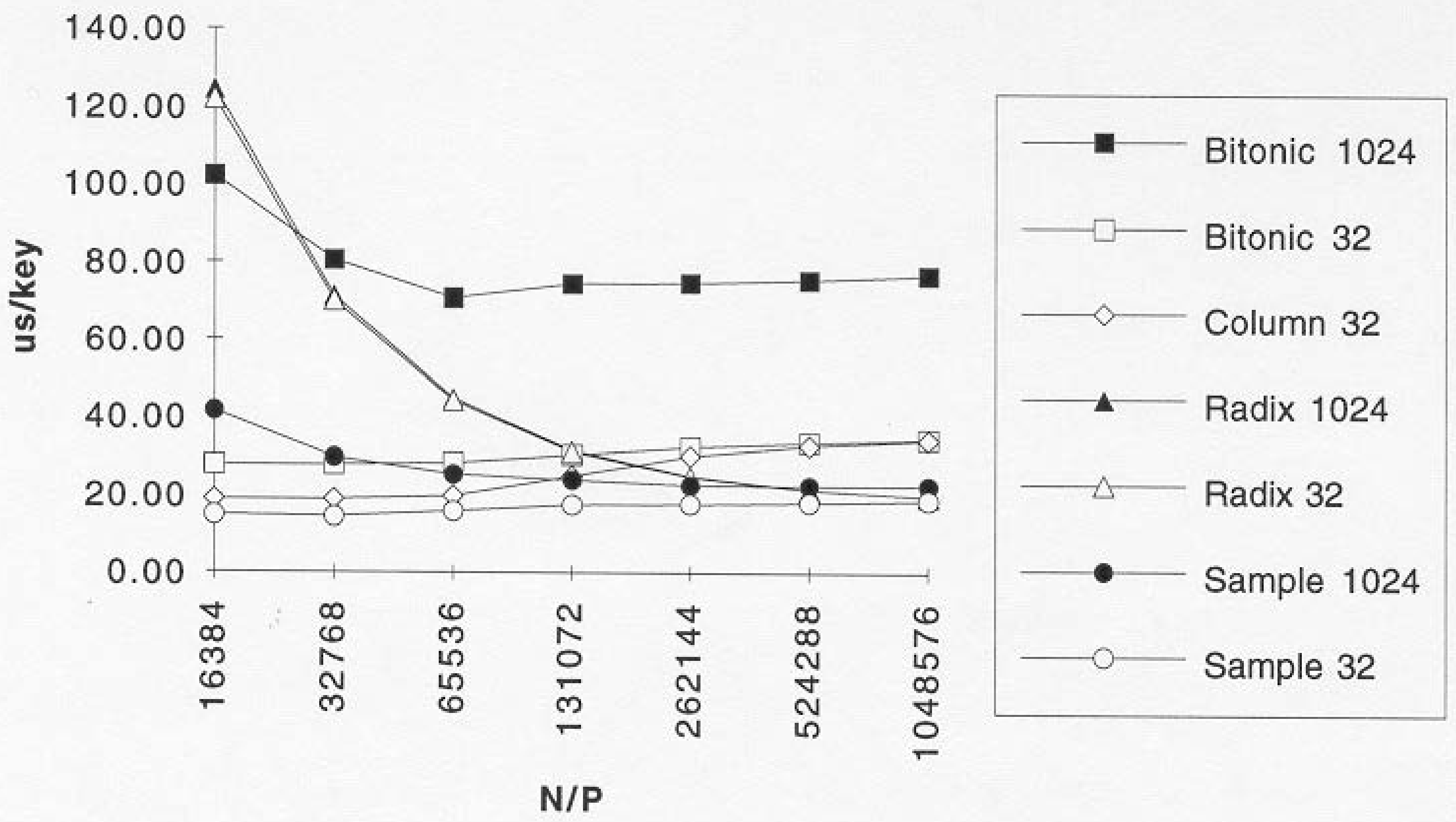
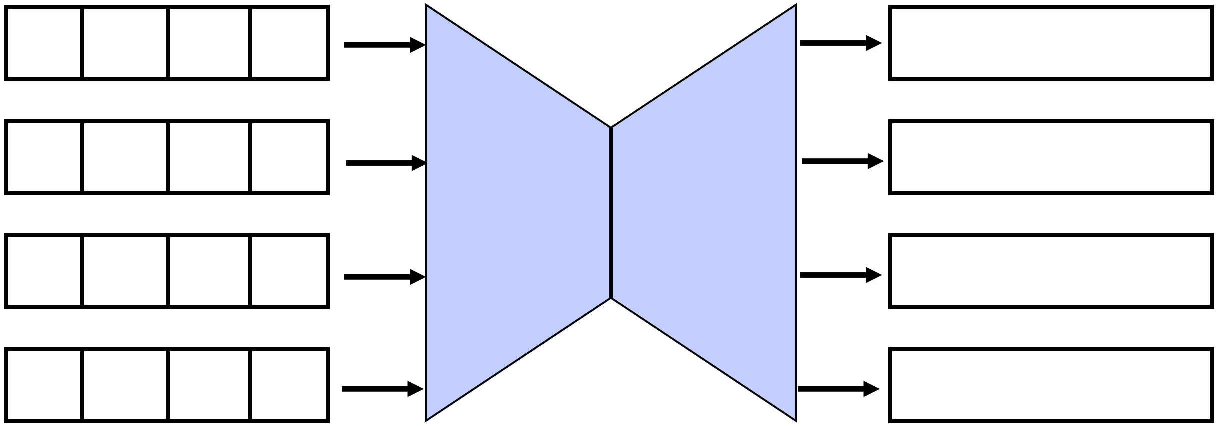


Figure 1.14 *Estimated execution time of four parallel sorting algorithms under LogP with the performance characteristics of the CM-5.*

Samplesort issues

- **Implementing the permutation**
  - What is the destination address of a given value? Two strategies:
    - » Send-to-queue operation
      - don't care, maintain queue at destination
    - » Compute unique destination for each value
      - planning cost: O(p) + 2pg + 2L
  - In what order should the values be sent?
    - » Global rearrangement defines a permutation, but piecewise implementation may yield poor performance



Samplesort issues

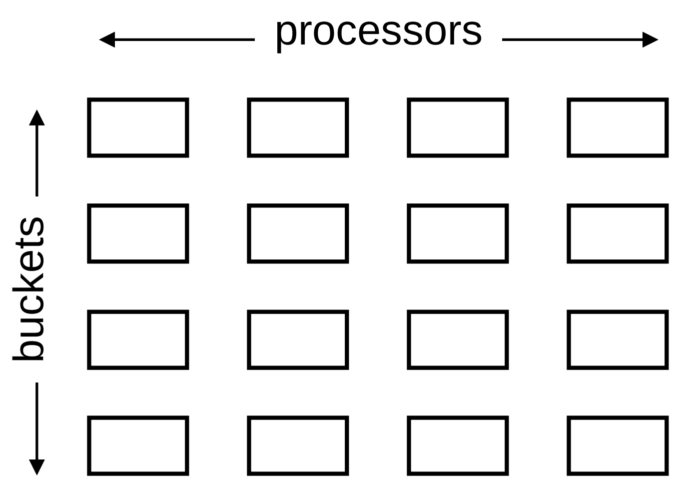
- **How to handle duplicate keys**
  - make each key unique
    - » (key, original index)
      - increases comparison cost and network traffic
  - random choice of possible destinations
    - » suppose p = 5 and splitters are

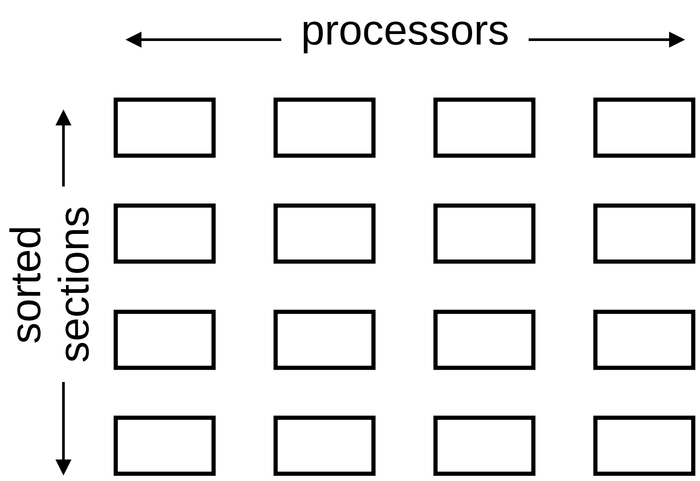
      10, 20, 20, 30

      where should we send key 20?
- **What about restoring load balance?**
  - Worst-case communication cost?

Two-phase sample sort

- **Objectives**
  - scramble input data to create a random permutation
  - highly supersample input to minimize skew





» Randomly distribute keys into p buckets

» Transpose buckets and processors

- expected bucket size N/p^2

» Local sort

» Proc 1 selects and broadcasts splitters

- oversampling ratio $k = N/p^2$

» Partition local keys into sorted sections according to splitters

- expected bucket size N/p^2

» Transpose sorted sections and processors

» Local p-way merge

Two-phase samplesort

1. Randomly distribute local keys into p local buckets
2. Transpose buckets and processors

---

3. Local sort
4. Processor 1 selects (p-1) splitters
5. Broadcast splitters

---

6. Local partitioning of values into p sorted sections
7. Transpose sorted sections and processors

---

8. Local p-way merge of sorted sections

$$C^{2\text{ph}}(N,p) = O\left(\frac{N}{p}\lg N\right) + 2\left(\frac{N}{p}\right)\cdot g + L + O\left(p\lg\left(\frac{N}{p}\right)\right) + 2p\cdot g + 3\cdot L$$