

# A FAST PROBABILISTIC PARALLEL SORTING ALGORITHM\*

Rüdiger Reischuk\*\*

Universität Bielefeld, Fakultät für Mathematik,  
4800 Bielefeld 1, West Germany

## Abstract

We describe a probabilistic parallel algorithm to sort  $n$  keys drawn from some arbitrary total ordered set. This algorithm can be implemented on a parallel computer consisting of  $n$  RAMs, each with small private memory, and a common memory of size  $O(n)$  such that the average run-time is bounded by  $O(\log n)$ . Hence for this algorithm the product of time and number of processors meets the information theoretic lower bound for sorting.

## 1. Introduction and known results

This paper deals with parallel algorithm for comparison problems. We are given a set of  $n$  keys drawn from a total ordered set of arbitrary size. The only operation that can be performed with the keys is the comparison of a pair. In the sequential case  $O(n)$  algorithms are known for selecting the  $k$ -smallest out of the  $n$  keys and  $O(n \log n)$  algorithms for sorting them, which meet (up to constant factors) the trivial lower bounds for these problems. Obviously these lower bounds also hold for parallel algorithms for the product of time and number of processors, but up to

now no parallel algorithm is known to the author that achieves these bounds.

In [V] Valiant proves an upper and lower bound  $\frac{n}{k} \log \log k + O(1)$  for finding the maximum of  $n$  elements with  $k$  processors. In his computation model he only counts the number of comparisons and doesn't care about additional necessary computations and flow of information. We will call this a parallel decision tree model. Obviously his lower bound holds quite generally. Hence we can conclude that for  $k = n$  processors with respect to sequential algorithms a speed-up of no more than  $O(n/\log \log n)$  instead of  $O(n)$  can be achieved for the problem of finding the maximum. In the same paper a parallel decision tree algorithm is described which sorts  $n$  keys with  $n$  processors in time  $O(\log n \log \log n)$ . Again the speed-up is only  $O(n/\log \log n)$ .

For another model of computation Preparata [P] obtains a parallel sorting algorithm which sorts  $n$  keys using  $n \log n$  processors in  $O(\log n)$  steps. His model is a single-instruction, multiple-data stream computer (SIMD) with random access capabilities to a common memory. Simultaneous reading in the same storage location is allowed, but no writing. Each processor performs arithmetic operations  $+, -, * [ / ], \lceil \log \rceil$  as well as the comparison of two keys in unit time. The algorithm recursively sorts smaller subsets and then merges pairs of ordered subsets in

\* Research supported by DFG-grant PA 248/1

\*\* Present address: IBM Research Laboratory,  
San José, California 95193

$O(\log \log n)$  steps together to determine the rank of each element. The above time bound includes the arithmetical computations, the number of comparisons and the number of read/write operations in the common memory, but doesn't take into consideration the problem of processor assignment when merging the subsets together. Of we do this merging costs  $O(\log n)$  steps which gives a total time bound  $O(\log^2 n / \log \log n)$ .

For this algorithm the speed-up for  $N = n \log n$  processors is  $O(N / \log N)$ , resp.  $O(N \log \log N / \log^2 N)$ .

For the same computation model [P] describes a second algorithm which uses  $n^{1+\alpha}$  processors and runs in time  $O(\frac{1}{\alpha} \log n)$  for any  $0 < \alpha \leq 1$ , which gives a speed-up of  $O(N^{\frac{1}{1+\alpha}})$ ,  $N = n^{1+\alpha}$ . The assignment of processors is easier in this algorithm and different processors don't read in a storage location at the same time.

Valiant's results show that for worst case time complexity not every problem can be solved  $n$  times faster if we use  $n$  instead of 1 processor. He also conjectures that the attainable speed-up for the selection problem is still smaller. The question arises whether a speed-up of order  $n$  can be achieved at least on the average, whether probabilism can improve parallel algorithms substantially. Obviously for selection or sorting probabilism doesn't help much in the sequential case. In the following we'll show that for both problems a speed-up of order  $n$  can be achieved by probabilistic parallel algorithms that use  $n$  processors.

We assume that the keys are distinct and have a linear order. This is no restriction since if we are given  $n$  arbitrary keys  $x_1, \dots, x_n$  replace  $x_i$  by  $(x_i, i)$  and define an order of the tuples by

$(x_i, i) < (x_j, j)$  iff  $x_i < x_j$  or  $x_i \leq x_j$  and  $i < j$ .

## 2. Selection in constant time

Theorem 1: There is a  $n$ -processor parallel decision tree algorithm to select the  $k$  smallest element out of a set of  $n$  keys ( $1 \leq k \leq n$ ), that finishes with probability greater than  $1 - 2^{-cn^{1/8}}$  in  $C$  steps for some constants  $c, C > 0$ .

Proof: The algorithm we describe is essentially a parallel version of the algorithm SELECT from [FR]. We expect the reader to be familiar with that paper.

PARSELECT ( $X, k, n$ ):

input: A set  $X$  of distinct keys  $x_1, \dots, x_m$ , a number  $k$ ,  $1 \leq k \leq m$ , the number of available processors  $n \geq m$ .

output: The  $k$  smallest element of  $X$ .

if  $m \leq \lfloor \sqrt{2n} \rfloor$  determine with  $\frac{m(m-1)}{2}$  comparisons - every element of  $X$  with every other - the  $k$  smallest element of  $X$  directly in one step.

if  $m > \lfloor \sqrt{2n} \rfloor$  do

1. Select probabilistically a subset  $S$  of size  $s := \lfloor \sqrt{m} \rfloor$ ; determine for each element of  $S$  its rank in  $S$  by comparing it with every other element of  $S$ .
2. Choose numbers  $1 \leq u(k) < v(k) < s$  and let  $y[z]$  be the  $u(k)$  [ $v(k)$ ] smallest element of  $S$ ; compare in two steps every element of  $X$  with  $y$  and  $z$ ; define

$A = \{x \in X \mid x \leq y\}$

$B = \{x \in X \mid y < x \leq z\}$

$C = \{x \in X \mid z < x\}$ ;

3. if  $|A| < k$  and  $|A| + |B| \geq k$   
     then PARSELECT (B, k-|A|, n)  
     if  $|A| \geq k$  then PARSELECT (A, k, n)  
     if  $|A| + |B| < k$   
         then PARSELECT (C, k-|A|-|B|, n)  
end PARSELECT.

From equations (13) and (14) in [FR] it follows that for

$$u(k) = \lfloor k \cdot \frac{s+1}{m+1} - d\sqrt{s} \rfloor$$

$$v(k) = \lfloor k \cdot \frac{s+1}{m+1} + d\sqrt{s} \rfloor$$

the probability that the  $k$  smallest element of  $X$  doesn't belong to  $B$  is less than  $O(\exp(-\frac{d^2}{2})/d)$ .

It remains to evaluate the size of  $B$ . The probability that  $B$  is of size  $\ell$  can be bounded by

$$\begin{aligned} P(\ell) &= \sum_{x=u(k)}^{m-(s-v(k))-\ell} \frac{\binom{x}{u(k)} \binom{\ell}{v(k)-u(k)} \binom{m-x-\ell}{s-v(k)}}{\binom{m}{s}} \\ &\leq \sum_x \frac{\binom{\ell}{v(k)-u(k)} \binom{m-\ell}{s-v(k)+u(k)}}{\binom{m}{s}} \\ &\leq m \frac{\binom{\ell}{g} \binom{m-\ell}{s-g}}{\binom{m}{s}} \text{ for } g := v(k)-u(k) = 2d\sqrt{s} \\ &\leq m \binom{s}{g} \left(1 - \frac{\ell}{m}\right)^{s-g} \left(\frac{\ell}{m}\right)^g \left(\frac{m}{m-s}\right)^g. \end{aligned}$$

The last term is less than  $\exp(\frac{s \cdot g}{m-2s})$ , hence for  $g = o(s)$  we get with an appropriate constant  $\alpha$

$$P(\ell) \leq \alpha \cdot m \binom{s}{g} \left(1 - \frac{\ell}{m}\right)^{s-g} \left(\frac{\ell}{m}\right)^g.$$

Using  $\binom{s}{g} \leq 2^{0(\log s) + s \cdot H(\frac{g}{s})}$

$[H(x) = x \log \frac{1}{x} + (1-x) \log \frac{1}{1-x}]$  and  $\log(1+x) \leq \frac{x}{1-x/2}$  for  $x > -1$  yields for  $g = o(s)$ :

$$\begin{aligned} \log P(\ell) &\leq O(\log m) + g \cdot \log \frac{s}{g} + s(1-\frac{g}{s}) \log \left(\frac{1}{1-\frac{g}{s}}\right) + \\ &\quad (s-g) \log \left(1 - \frac{\ell}{m}\right) + g \log \frac{\ell}{m} \\ &\leq O(\log m) + g \cdot \log \frac{s \cdot \ell}{g \cdot m} + s \log \left(1 + \frac{g}{s-g}\right) + \\ &\quad (s-g) \log \left(1 - \frac{\ell}{m}\right) \\ &\leq O(\log m) + g \cdot \log \frac{s \cdot \ell}{g \cdot m} + \frac{sg}{(s-g) \ln 2} - \\ &\quad \frac{(s-g)\ell}{m \ln 2} \\ &\leq O(\log m) + g \cdot O(\log m) - \theta\left(\frac{\ell}{s}\right) \\ &= O(\log m) + O(d \cdot m^{1/4} \cdot \log m) - \theta\left(\frac{\ell}{m^{1/2}}\right). \end{aligned}$$

This gives for  $d = m^{1/8}$  and  $\ell \geq m^{15/16}$

$$P(\ell) \leq 2^{-\theta(m^{7/16})}.$$

Therefore the probability that in 1) we make a bad choice, that means the  $k$  smallest element of  $X$  isn't in  $B$  or  $B$  is bigger than  $m^{15/16}$ , is at most  $\exp(-\theta(m^{1/4}))$ . This means that after having executed steps 1) and 2)  $11$  times with probability  $1 - 2^{-\theta(m^{1/4})}$  the  $k$  smallest element of the original set  $X$  belongs to a subset  $B$  of  $X$  with size less than  $m^{(15/16)^{11}} < m^{1/2}$  and we can find this element in  $B$  by one more step.

Corollary 1: In the parallel decision tree model sorting  $n$  keys with  $n$  processors can be performed in average time  $O(\log n)$ .

Proof: Use PARSELECT to find the median in average time bounded by a constant. Compare every key with the median to get two subsets  $S, G$  of those elements which are smaller [greater] than the median. Recursively sort  $S$  and  $G$  in parallel with  $\frac{n}{2}$  processors each. If  $T(n)$  denotes the expected number of comparison to sort  $n$  keys with  $n$  processors we get the recursion formula

$$T(n) \leq T\left(\frac{n}{2}\right) + \text{const.}$$

which implies  $T(n) \leq O(\log n)$ .

### 3. Fast probabilistic parallel sorting

We now describe a parallel sorting algorithm for a more realistic computation model than the parallel decision tree.

The parallel computer consists of  $n$  processors  $p(1), \dots, p(n)$  and a common memory of size  $O(n)$ . Each processor is a RAM with a constant number of private registers and has random access to the common memory. We allow simultaneous reading in the same storage location of the common memory by different processors, but no simultaneous writing. The arithmetic instruction set contains  $+, -, *, [ / ]$ . Each processor needs one unit of time to perform an arithmetic operation or to compare two keys. This implies that functions like  $\lfloor \sqrt{n} \rfloor$ ,  $\lfloor \log n \rfloor$ ,  $2^n$  all can be computed in  $O(\log n)$  steps.

Suppose we are given a set  $Y = \{y_1, \dots, y_n\}$  of  $n$  distinct keys which at the beginning are stored in the common memory in an array  $k = k(1), \dots, k(n)$ ,  $k(i) = y_i$ . This sequence of  $n$  keys will be rearranged to get a partition  $k = k^1, k^2, \dots, k^N$  into subsequences called boxes such that for  $i < j$  every element of box  $k^i$  is less than every element of  $k^j$ .

Such a partition of  $k$  is uniquely defined by the indices of the first and last element of each box. These indices are stored in arrays  $L = l(1), \dots, l(n)$  and  $U = u(1), \dots, u(n)$  in the following way:

if key  $k(i)$  belongs to box  $k^j = k(l+1), \dots, k(u)$  then  $l(i) = l+1$  and  $u(i) = u$ . At the beginning we set  $l(i) := 1$  and  $u(i) := n$  for all  $i$ .

Now the global structure of the sorting algorithm is the following:

1. Select a subset  $S$  of  $Y$  of size  $\lfloor \sqrt{n} \rfloor$  at random and sort  $S$  by comparing every pair of keys in  $S$ .

With the help of this ordered subset  $Y$

can be partitioned into  $\lfloor \sqrt{n} \rfloor + 1$  boxes  $B_0, \dots, B_{\lfloor \sqrt{n} \rfloor}$  where  $B_i$  contains those keys that are bigger than the  $i$ -th smallest but not bigger than the  $(i+1)$ -smallest element of  $S$ .

2. By binary insertion determine in parallel for each key of  $Y$  into which box it falls.
3. Rearrange the sequence  $k(1), \dots, k(n)$  in such a way that all elements of  $B_i$  proceed all elements of  $B_j$  for  $i < j$ .
4. Recursively in parallel for each  $j$  sort the subsequence of  $k$  that contains the elements of box  $B_j$ .

It will be shown that with great probability the size of every box  $B_i$  is less than  $O(\sqrt{n} \log n)$ . Thus on the average in the course of the recursion the problem size decreases fast.

While the algorithm is quite simple, the problem is to find a fast implementation for a parallel computer of the above type.

The following notation will be used for a subset  $I \subset \{1, \dots, n\}$ :

for  $i \in I$   $p(i)$ : <commands> ,

this means that in parallel each processor  $p(i)$  with  $i \in I$  carries out the commands.

The algorithm uses two subroutines which will be described and analysed beforehand. In the first one a key is selected from a subsequence of  $k$  at random and stored in an array  $s = s(1), \dots, s(n)$ .

SEL( $l+1, u, s$ )

comment: processor  $p(l+1)$  selects a key  $s(l+1)$  from  $k(l+1), \dots, k(u)$  at random.

processor  $p(\ell+1)$  :

1. choose a  $2^{\lceil \log(u-\ell) \rceil}$  - digit binary number  $N$  at random.
2.  $m := N \bmod (u-\ell) = N - (u-\ell) \lfloor \frac{N}{u-\ell} \rfloor$ .
3.  $s(\ell+1) := k(\ell+1+m)$ .

end SEL

For each key among  $k(\ell+1), \dots, k(u)$  the probability to be chosen is either

$$\alpha = \lfloor \frac{2^{\lceil \log(u-\ell) \rceil}}{u-\ell} \rfloor / 2^{\lceil \log(u-\ell) \rceil} \text{ or}$$

$\beta = \lceil \frac{2^{\lceil \log(u-\ell) \rceil}}{u-\ell} \rceil / 2^{\lceil \log(u-\ell) \rceil}$  which differ by at most  $(u-\ell)^{-2}$ . Since  $x^{-1} - x^{-2} \geq (x+2)^{-1}$  for  $x \geq 2$  it follows

Proposition 1: For each key among  $k(\ell+1), \dots, k(u)$  the probability to be chosen by  $\text{SEL}(\ell+1, u, s)$  is at least  $(u-\ell+2)^{-1}$  if  $u-\ell \geq 2$ .

If each processor can make a 0/1-random choice in one step the runtime of  $\text{SEL}(\ell+1, u, s)$  is bounded by  $O(\log(u-\ell))$ .

The next procedure computes all partial sums of a sequence of  $m$  numbers in  $O(\log m)$  steps.

ALLSUM( $\ell+1, u, c, d$ )

comment: processors  $p(\ell+1), \dots, p(u)$  compute all sums  $\sum_{j=\ell+1}^i c(j)$  for  $i = \ell+1, \dots, u$  and store the results in  $d(\ell+1), \dots, d(u)$ .

1. for  $i \in \{\ell+1, \dots, u\}$   $p(i)$  :  
 $d(i) := c(i)$ .

2. for  $k = 1, \dots, \lceil \log(u-\ell) \rceil$  do  
for  $i \in \{1, \dots, \lfloor (u-\ell)/2^k \rfloor\}$   
 $p(\ell+2^k \cdot i)$  :  
 $d(\ell+2^k \cdot i) := d(\ell+2^k \cdot i) +$   
 $d(\ell+2^k \cdot i - 2^{k-1})$ .

end

comment: now for  $\ell+1 \leq j \leq u$  with  $j = \ell + 2^k \cdot i$ ,  $0 \leq k \leq \lceil \log(u-\ell) \rceil$  and  $i \equiv 1 \pmod{2}$  holds  
 $d(j) = d(\ell + 2^k \cdot i) =$   
 $\sum_{x=1}^{2^k} c(\ell + 2^k \cdot (i-1) + x)$ .

3. for  $i \in \{\ell+1, \dots, u\}$   $p(i)$  :  
 $q(i) := d(i)$ ,  
 $e(i) := \ell$   
 $d(i) := 0$ ,  
for  $k = \lceil \log(u-\ell) \rceil, \dots, 0$  do  
if  $e(i) + 2^k \leq i$  then  
 $e(i) := e(i) + 2^k$ ,  
 $d(i) := d(i) + q(e(i))$ .

end

end ALLSUM

We now describe a recursive procedure PPSORT to sort a box  $B^{\ell+1, u} = k(\ell+1), \dots, k(u)$  with processors  $p(\ell+1), \dots, p(u)$ . As input parameters we need only  $\ell+1$  and  $u$ , the lower and upper boundary of  $B^{\ell+1, u}$ . Global parameters for this procedure are the sequence  $k$  of keys and the array  $L$  and  $U$  for the partition of  $k$ . To store intermediate results we use some additional arrays  $s, \tilde{s}, c, d, D, g_1, G_1, g_2, G_2$  each of length  $n$ . For each of these arrays during PPSORT( $\ell+1, u$ ) processors  $p(\ell+1), \dots, p(u)$  only access array elements with indices between  $\ell+1$  and  $u$ .

PPSORT ( $\ell+1, u$ ) :

if  $u-\ell < 16$  sort  $k(\ell+1), \dots, k(u)$  with  $p(\ell+1), \dots, p(u)$  by pairwise comparisons

if  $u-\ell \geq 16$  then do

1. for  $i \in \{\ell+1, \dots, u\}$   $p(i)$  :  
 compute  $w := \lfloor \sqrt{u-\ell} \rfloor$ ,  $v := \lfloor \frac{u-\ell}{w} \rfloor$ ,  
 $z := 2^{\lfloor \log w \rfloor + 2}$  and store  $w, v, z$   
 in the private memory.

comment: in step 2 to 5 a subset  $S$  of  $\{k(\ell+1), \dots, k(u)\}$  of size  $w$  will be selected at random and sorted.

2. for  $j \in \{0, \dots, w-1\}$  do in parallel  
 $\text{SEL}(\ell+jv+1, \ell+(j+1)v+(i+1)v, c, d)$  .

3. for  $i, j \in \{0, \dots, w-1\}$   $p(\ell+iv+j+1)$  :  
if  $s(\ell+iv+1) \geq s(\ell+jv+1)$   
then  $c(\ell+iv+j+1) := 1$   
else  $c(\ell+iv+j+1) := 0$

4. for  $i \in \{0, \dots, w-1\}$  do in parallel  
 $\text{ALLSUM}(\ell+iv+1, \ell+(i+1)v, c, d)$  .

comment:  $d(\ell+(i+1)v)$  is the rank of  $s(\ell+iv+1)$  in  $S$  .

5. for  $i \in \{0, \dots, w-1\}$   $p(\ell+iv+1)$  :  
 $\tilde{s}(\ell+d(\ell+(i+1)v)) := s(\ell+iv+1)$  .

comment:  $\tilde{s}(\ell+1), \tilde{s}(\ell+2), \dots, \tilde{s}(\ell+w)$  is the ordered sequence of keys in  $S$  .

comment: in step 6 to 7 we'll determine into which box  $B_j^{\ell+1, u}$  each key of  $B^{\ell+1, u}$  falls, where

$$B_j^{\ell+1, u} := \{k(i) \mid \ell+1 \leq i \leq u, \tilde{s}(\ell+j) < k(i) < \tilde{s}(\ell+j+1)\} \text{ for } 1 \leq j < w,$$

$$B_0^{\ell+1, u} := \{k(i) \mid \ell+1 \leq i \leq u, k(i) < \tilde{s}(\ell+1)\}$$

$$B_w^{\ell+1, u} := \{k(i) \mid \ell+1 \leq i \leq u, \tilde{s}(\ell+w) \leq k(i)\}$$

6. for  $i \in \{\ell+1, \dots, u\}$   $p(i)$  :  
 $c(i) := 0$  .

7. for  $k = \lfloor \log w \rfloor, \dots, 0$  do  
for  $i \in \{\ell+1, \dots, u\}$   $p(i)$  :  
if  $c(i) + 2^k \leq w$  and  $\tilde{s}(\ell+c(i)+2^k) \leq k(i)$   
then  $c(i) := c(i) + 2^k$

end

comment: one verifies easily that  $c(i)$  is the index of that box into which key  $k(i)$  falls; next compute the size of the boxes  $B_j^{\ell+1, u}$  and the relative position of each element in its box.

8. for  $i \in \{\ell+1, \dots, u\}$   $p(i)$  :  
 $d(i) := z^{c(i)}$  .

9.  $\text{ALLSUM}(\ell+1, u, d, D)$  .

comment: if  $D(s) = \sum_{i=\ell+1}^s d(i)$ ,  $s \in \{\ell+1, \dots, u\}$ , is expanded to the base  $z$ , that means  $D(s) = \sum_{j=0}^w h(j, s) z^j$  with  $0 \leq h(j, s) \leq u-\ell < z$ , then the coefficient  $h(j, s)$  counts the number of elements among  $k(\ell+1), \dots, k(s)$  that fall into box  $B_j^{\ell+1, u}$  .

10. for  $j \in \{0, \dots, w\}$   $p(\ell+1+j)$  :

$$g_1(\ell+1+j) := h(j, u) = \lfloor \frac{D(u)}{z^j} \rfloor - z \lfloor \frac{D(u)}{z^{j+1}} \rfloor .$$

11. ALLSUM( $\ell+1, \ell+1+w, g_1, G_1$ ).

comment:  $g_1(\ell+1+j), G_1(\ell+1+j) = \sum_{\ell=0}^j g_1(\ell+1+k)$  is the number of elements of  $B^{\ell+1, u}$  that fall into  $B_j^{\ell+1, u}$ , resp. into  $B_0^{\ell+1, u} \cup \dots \cup B_j^{\ell+1, u}$ .

12. for  $i \in \{\ell+1, \dots, u\}$   $p(i)$ :

$g_2(i) := h(c(i), i) =$

$\lfloor \frac{D(i)}{z \cdot c(i)} \rfloor - z \lfloor \frac{D(i)}{z \cdot c(i) + 1} \rfloor,$

if  $c(i) = 0$

then  $G_2(i) := g_2(i)$

else  $G_2(i) := G_1(\ell+1+c(i)-1) + g_2(i),$

$s(\ell+G_2(i)) := k(i),$

$k(i) := s(i).$

comment:  $G_2(i)$  counts the number of elements among  $k(\ell+1), \dots, k(i)$  that belong to a box  $B_j^{\ell+1, u}$  with index  $j$  less than or equal to  $c(i)$ , the index of that box where  $k(i)$  belongs to; therefore  $G_2(\ell+1), \dots, G_2(u)$  is a permutation of  $\{1, \dots, u-\ell\}$  and after rearranging the array  $k$  the elements of  $B_j^{\ell+1, u}$  are stored in

$k(\ell+G_1(\ell+1+j-1)+1), \dots, k(\ell+G_1(\ell+1+j)).$

13. for  $i \in \{\ell+1, \dots, u\}$   $p(i)$ :

if  $c(i) = 0$

then  $\ell(\ell+G_2(i)) := \ell$

else  $\ell(\ell+G_2(i)) := \ell + G_1(\ell+c(i)),$

$u(\ell+G_2(i)) := \ell + G_1(\ell+1+c(i)).$

comment: in step 13 lower and upper boundaries for the boxes  $B_j^{\ell+1, u}$  are computed.

14. for  $j \in \{0, \dots, w\}$  do in parallel

if  $j = 0$

then PPSORT( $\ell+1, \ell+G_1(\ell+1)$ ),

else PPSORT( $\ell+1+G_1(\ell+j), \ell+G_1(\ell+1+j)$ ).

end PPSORT

The correctness of this algorithm can easily be seen. Note that if each processor  $p(i)$  "knows" his label  $i$  then all processors can be programmed equally. It remains to estimate a time bound for the algorithm. For  $u-\ell = n$  we have  $w = \lfloor \sqrt{n} \rfloor$  and for  $v = \lfloor \frac{n}{\lfloor \sqrt{n} \rfloor} \rfloor$  holds

$\lfloor \sqrt{n} \rfloor \leq v \leq \lfloor \sqrt{n} \rfloor + 2$ . Let  $T(n)$  be the average time the parallel computer needs to sort  $n$  keys  $k(\ell+1), \dots, k(u)$  with processors  $p(\ell+1), \dots, p(u)$ . If  $n_j, 0 \leq j \leq \lfloor \sqrt{n} \rfloor$ , denotes the size of the box  $B_j^{\ell+1, u}$  then

$$T(n) = O(\log n) + \max_j T(n_j)$$

if we assume  $T(n)$  to be increasing with  $n$ .

The probability that for some  $j$   $n_j$  exceeds  $\alpha \cdot \sqrt{n}$  can be bounded by the following estimation:

Take any subsequence of length  $b > \lfloor \sqrt{n} \rfloor$  of the ordered sequence of keys and let  $b_i (1 \leq i \leq w+1)$  be the number of those keys of this subsequence that belong to the  $i$ -th interval  $I_i$  of  $k(\ell+1), \dots, k(u)$ ,  $I_i = k(\ell+(i-1)v+1), \dots, k(\ell+iv)$  for  $1 \leq i \leq w$ ,  $I_{w+1} = k(\ell+wv+1), \dots, k(u)$ .

It holds  $b_{\lfloor \sqrt{n} \rfloor + 1} \leq u - (l + wv) =$

$$n - \lfloor \sqrt{n} \rfloor \lfloor \frac{n}{\lfloor \sqrt{n} \rfloor} \rfloor \leq \lfloor \sqrt{n} \rfloor .$$

The probability not to hit any of these elements when selecting from each interval  $I_1, \dots, I_w$  one element at random is by proposition 1 at most

$$p = \max_{b_1, \dots, b_{w+1}} \prod_{i=1}^w (1 - \frac{b_i}{v+2}) .$$

Taking the logarithm yields:

$$\begin{aligned} \ln P &= \max_{b_i} \sum_{i=1}^w \ln(1 - \frac{b_i}{v+2}) \leq \max_{b_i} \sum_{i=1}^w -\frac{b_i}{v+2} \\ &\leq \max_{b_i} \frac{-b + b_{w+1}}{v+2} \leq \frac{-b + \lfloor \sqrt{n} \rfloor}{\lfloor \sqrt{n} \rfloor + 4} . \end{aligned}$$

For  $b \geq \alpha \cdot \sqrt{n}$  we get  $\ln P \leq -(\alpha - o(1))$  or  $P \leq O(e^{-\alpha})$ .

For  $\alpha = 3 \ln n$  the total probability that some  $n_j$  exceeds  $\alpha \sqrt{n}$  is less than  $(n - \alpha \sqrt{n} + 1) O(e^{-3 \ln n}) \leq O(\frac{1}{n^2})$ .

This gives

$$\begin{aligned} T(n) &\leq O(\log n) + \\ &\text{Prob}(\max_j \leq 3 \ln n \sqrt{n}) T(3 \ln n \sqrt{n}) + \\ &\text{Prob}(\max_j n_j > 3 \ln n \sqrt{n}) T(n - \lfloor \sqrt{n} \rfloor + 1) \leq \\ &O(\log n) + T(3 \ln n \sqrt{n}) + O(\frac{1}{n^2}) T(n - \lfloor \sqrt{n} \rfloor + 1) . \end{aligned}$$

By induction one shows  $T(n) \leq O(\log n)$ .

Thus we have proved:

**Theorem 2:**  $n$  keys can be sorted by a  $n$ -processor parallel random access computer in average time  $O(\log n)$ .

## References

- [P] Preparata, *New parallel-sorting schemes*, IEEE Trans. on Comp., vol. c-27, No. 7, 669-773, July 1978.
- [V] L.G. Valiant, *Parallelism in comparison problems*, SIAM J. Comput., vol. 4, 348-355, September 1975.
- [FR] R. Floyd and R. Rivest, *Expected Time Bounds for Selection*, Com. ACM, vol. 18, 165-172, March 1975.