SIAM J. COMPUT.
Vol. 14, No. 3, August 1985

ON PARALLEL SEARCHING*

MARC SNIR†

Abstract. We investigate the complexity of searching a sorted table of n elements on a synchronous, shared memory parallel computer with p processors. We show that $\Omega(\lg n - \lg p)$ steps are required if concurrent accesses to the same memory cell are not allowed, whereas $O(\lg n/\lg p)$ steps are sufficient if simultaneous reads are allowed. The lower bound is valid even if only communication steps are counted, and the computational power of each processor is not restricted. In this model, $\Theta(\sqrt{\lg n})$ steps are required for searching when the number of processors is unbounded. If the amount of information that a memory cell may store is restricted, then the time complexity for searching with an unbounded number of processors is $\Theta(\lg n/\lg \lg n)$. If the amount of information a processor may hold is also restricted, then an $\Omega(\lg n)$ lower bound holds. These lower bounds are first proven for comparison-based algorithms; it is next shown that comparison-based algorithms are as powerful as more general ones in solving problems defined in terms of the relative order of the inputs.

Key words. parallel algorithms, parallel computations

1. Introduction.

With the advance in microelectronics it becomes feasible to build parallel machines with thousands of cooperating processors. Yet, practice indicates that a thousandfold increase in raw computational power does not increase performance by the same amount. There are two main reasons for that. The first one is that not every problem admits an efficient parallel solution. The second one is that not every parallel algorithm can be mapped efficiently onto a realistic parallel computer architecture. Work sharing between many processors generates significant overheads for communication of data and coordination. The study in parallel complexity is dedicated, to a large extent, to the understanding of these two phenomena.

One useful model for the study of parallel computations is that of a *paracomputer*. It consists of many identical autonomous processors, each with its own local memory and its own program. In addition, the machine has a shared memory and each processor can in one step access any cell in shared memory.

We obtain successively weaker models by varying the assumptions concerning simultaneous accesses to shared memory:

(1) *Concurrent Read, Concurrent Write* (*CRCW*). Both simultaneous reads and simultaneous writes to the same memory cell are allowed. The effect of simultaneous actions by the processors is as if the actions occurred in some serial order (for other possible definitions of CRCW, see [2]).

(2) *Concurrent Read, Exclusive Write* (*CREW*). Simultaneous reads are allowed but a processor can modify a shared memory cell only if it has exclusive access to it.

(3) *Exclusive Read, Exclusive Write* (*EREW*). Simultaneous accesses to the same shared memory cell are not allowed.

This model can be further weakened in two ways: One can restrict the number of shared memory cells. One can also restrict the set of processors that have read or write access to each memory cell. If there is a unique processor that has read access and a unique processor that has write access to each shared memory cell, then each

* Received by the editors June 15, 1982, and in final revised form February 21, 1984. This work was supported in part by the National Science Foundation under grant MCS-8203307.

† Courant Institute of Mathematical Sciences, New York University, New York, New York 10012. Present address, Department of Computer Science, The Hebrew University of Jerusalem, Jerusalem 91904, Israel.

memory cell represents a unidirectional link that connects two processors. We speak then of an *ultracomputer*.[1]

In these models communication both of data and of control information is done through the shared memory. Thus, studying the relative power of these models is tantamount to studying the effect of constraints on communication and coordination on computational power.

We consider the problem of searching a key within a sorted list of n keys. The binary search algorithm solves this problem in (optimal) sequential time $O(\lg n)$. This can be generalized to a "$(p+1)$-ary" search algorithm that solves the problem in $O(\log_{p+1} n)$ steps with p processors. At each step p comparisons are done that split the list into $p+1$ equal length segments, and the search proceeds recursively within the unique segment that may contain the key. This algorithm is optimal, so that p processors speed up searching by a factor of $\lg(p+1)$ only: Searching does not admit an efficient parallel solution.

Consider now the problem of implementing this parallel algorithm. It turns out that the speedup can be achieved only if one item of information can be broadcast to all processors in constant time. On the other hand we show that in the EREW model, where an item of information can be accessed by one processor only at a time, searching requires at least $\Omega(\lg n - \lg p)$ steps. Note that no transmission of data is required for parallel search, but the processors need to coordinate the search at each iteration. It turns out that the time spent in coordinating the processors offsets exactly the gain obtained from simultaneous table look-ups.

The $\Omega(\lg n - \lg p)$ lower bound is valid even if each processor may in one step do any amount of local processing or transfer any amount of information. The only restrictions are that at each step a processor may read or write at a unique location in memory.

This result settles the problem of the relative power of the different shared-memory machine models. It is known that a p-processor machine which supports concurrent accesses to the same location in memory can be simulated by a p-processor machine with no concurrent accesses to the same location in memory with a $\lg p$ time penalty [5]. Our result shows that this simulation is optimal.

Under the same assumptions we prove that the time complexity of searching with an unbounded number of processors is $\Theta(\sqrt{\lg n})$.

The computational model is very strong since there are no restrictions on the amount of information that can be transmitted in one step. This is remedied by assuming that a memory cell may contain a unique input value, and that inputs are atomic entities, so that an input symbol cannot be used to encode the values of several inputs. A memory cell may also store a symbol taken from a small domain of internal values. In this model the time complexity of searching with an unbounded number of processors is $\Theta(\lg n/\lg \lg n)$. Finally, if we impose a similar restriction on the local memory of each of the processors, then an $\Omega(\lg n)$ lower bound is valid, independently of the number of processors.

The $\Omega(\lg n)$ lower bound on search implies a similar lower bound for the insertion problem on a shared memory parallel machine with no concurrent access to the same memory location. This settles an open problem posed by Borodin and Hopcroft in [2].

The lower bounds for searching with an unbounded number of processors are proven for comparison-based computations. We also prove that comparison-based

[1] The terms paracomputer and ultracomputer are taken from [10], but are used here in a slightly different meaning.

algorithms are as efficient as more general ones in solving problems that are defined in terms of the relative order of the inputs. We first show that if such a problem can be solved by comparisons for inputs taken from a subset of values where every possible permutation of the inputs occurs, then it can be solved by comparisons, using the same resources, for any input. We next prove that given a paracomputer, one can build a sufficiently large set of input values where the behavior of each processor at each step depends only on the relative order of the pairs of inputs it has access to, but not on their actual values. The last result is proven by an application of Ramsey's theorem.

The remainder of this paper is organized as follows. The implementation of the "$(p+1)$-ary" searching algorithm is discussed in the next section. In § 3 we prove the $\Omega(\lg n - \lg p)$ lower bound for a simplified version of the searching problem. This is followed in § 4 by a description of an $O(\sqrt{\lg n})$ algorithm for searching in the EREW model with $O(n)$ processors. In § 5 we present the reduction of general paracomputer computations to computations using only comparisons. This reduction is used in § 6 to prove the $\Omega(\sqrt{\lg n})$ lower bound for searching with an unrestricted number of processors. In § 7 we examine the complexity of searching in restricted paracomputer models. Conclusions and open problems are brought in the last section.

2. Parallel searching in the CREW model.

The search problem has several variants, all of which have essentially the same complexity. For sake of simplicity we consider the following version.

Range search problem (for a table of size n). Given $n+1$ distinct inputs $x_1, \cdots, x_n, y$ such that $x_1 < \cdots < x_n$, find the index i such that $x_i < y < x_{i+1}$. (By definition $x_0 = -\infty$ and $x_{n+1} = \infty$.)

We first determine the complexity of searching in the CRCW and CREW models of parallelism.

THEOREM 2.1. *The range search problem for a table of size n can be solved on a CREW machine with p processors in time* $O(\lg(n+1)/\lg(p+1))$.

Proof. The algorithm used is the obvious extension of binary search to p processors, namely $(p+1)$-ary search: p keys are chosen that divide the list of keys into $p+1$ intervals of roughly equal length. These keys are compared in parallel to the searched key. The comparisons locate the searched key within one of the subintervals, and the search proceeds recursively within this subinterval. At each iteration the length of the list is decreased by a factor of $p+1$, so that the search ends in $\lg_{p+1}(n+1)$ iterations. It remains to be shown that each iteration can be implemented in constant time, without concurrent writes.

Note that the searched key is located within the ith subinterval iff the outcomes of the comparisons made at processor $i-1$ and at processor i are different. Each processor can in constant time match the outcome of the comparison it performed against the outcome of the comparison performed by its right neighbor. The unique processor that detects the subinterval containing the searched key then updates the shared information on the search interval. All the processors next read this information, and proceed to search within the new interval. The complete algorithm is given below. The syntactic construct

```
for $i$ in $S$ pardo $P(i)$ odrap
```

indicates parallel execution of the statements $P(i)$ for each value of i in the set S. Variables declared within a parallel loop are private to the processor executing this instance of the loop.

```
SEARCH ($y$, $\vec{x}$, bot, top)
/*Search for key $y$ in sorted list $x_{\text{bot}}, \cdots, x_{\text{top}}$.
We assume that $x_{\text{bot}} < y < x_{\text{top}}$.
Initially bot $= 0$, top $= n+1$, $x_{\text{bot}} = -\infty$, $x_{\text{top}} = \infty$*/
cons
    $p$;            /*Number of processors*/
var
    newbot, newtop;
    $c_1, \cdots, c_{p+1}$; /*Vector storing outcomes of comparisons*/
begin
    $c_{p+1} := 1$;
while (top > bot + 1) do
  for $j$ in $[1, p]$ pardo
    var
        $i$;
    /*Compute index of key to be compared*/
    $i :=$ bot $+ j*$(top-bot)$/(p+1)$;
    /*Compare and store outcome*/
    $c_j :=$ if $y > x_i$ then 0 else 1
    odrap;
    /*Compute next interval*/
    ⟨newbot, newtop⟩ := ⟨bot, (top-bot)$/(p+1)$⟩;
  for $j$ in $[1,p]$ pardo
    if $c_j < c_{j+1}$ then
      ⟨newbot, newtop⟩ := ⟨bot $+ j*$(top-bot)$/(p+1)$,
                          bot $+ (j+1)*$(top-bot)$/(p+1)$⟩
    odrap;
    ⟨bot, top⟩ := ⟨newbot, newtop⟩
  od;
return (bot)
end
```

Note that the full power of concurrent reads is not needed to implement this algorithm. It is sufficient to have a shared memory machine with broadcasting ability: One (fixed) processor is able to broadcast in constant time one item of information to all the other processors.

Parallel search in the continuous case was studied by Gal and Miranker in [6], where a parallel version of the bisection algorithm for root finding is given (the problem of processors coordination is ignored there). An adversary argument is used there to show that the policy of splitting at each stage the interval of possible values into equal length subintervals is optimal. The same argument applies to the discrete case, and implies the following result.

THEOREM 2.2. *Let P be a parallel algorithm that solves the range search problem for a table of size n, such that at each step at most p values from the table are accessed. Then the algorithm executes in the worst case at least* $\lg(n+1)/\lg(p+1)$ *steps.*

Thus the complexity of searching by comparisons a table of size n with p processors is $\Theta(\lg(n+1)/\lg(p+1))$, if concurrent reads are supported, both for parallel machines that support concurrent writes and for parallel machines that do not support concurrent writes.

Concurrent reads may occur at several places in the algorithm given in Theorem 2.1:

(i) If the search interval is small, then a key from the searched table may be accessed by more than one processor.

(ii) All the processors share the searched key.

(iii) After each iteration, all the processors read concurrently the bounds of the new search interval.

The first type of concurrent reads may be avoided by more careful programming. The concurrent accesses to the searched key may be avoided by initially distributing it to all the processors in $O(\lg p)$ steps. If concurrent reads are not supported, then the broadcasting of the next search interval at the end of each iteration will require $\Omega(\lg p)$ steps, and the running time of the algorithm for fixed p will be $\Omega(\lg n - \lg p)$, with practically no gain obtained from parallelism. It turns out that this is indeed the best performance that can be achieved for searching on a parallel machine that does not support concurrent accesses to the same location in memory: $\Omega(\lg n - \lg p)$ steps are required, even if we do not account for the distribution of the searched key.

3. Lower bounds for discrete root finding.

We shall prove the $\Omega(\lg n - \lg p)$ lower bound for the particular case of the range finding problem where the value of the searched key is fixed. This restricted problem can be reformulated as follows. We denote by $\vec{w}_i^n$ the binary sequence consisting of i zeros followed by $n-i$ ones. The index n will be omitted when it can be inferred from the context.

Discrete root finding (for a sequence of length n). Given a monotonic binary sequence $\langle x_1, \cdots, x_n\rangle$ count the number of zeros in it, i.e. find the index i such that $\langle x_1, \cdots, x_n\rangle = \vec{w}_i^n$.

We first give a more formal definition of the paracomputer model. A paracomputer consists of p processors $P_1, \cdots, P_p$, q registers $R_1, \cdots, R_q$, an input set X, a set of processor states S, a set of register symbols $V \supset X$, a time bound T, a subset of n registers $I_1, \cdots, I_n$ that are used for input, and a subset of k registers $O_1, \cdots, O_k$ that are used for output.

With each processor P_i are associated the following (partial) functions.

$\alpha_i : S \to \{1, \cdots, q\} \times \{R, RW\}$—the access function. The first component of $\alpha_i(s)$ yields the index of the register accessed by P_i when in state s; the second component indicates whether the access is a read (R) or a read and write (RW) operation. We assume w.l.g. that each processor accesses at each step a shared memory location.

$\omega_i : S \to V$—the write function. $\omega_i(s)$ yields the symbol written by P_i, when in state s.

$\delta_i : S \times V \to S$—the state transition function. If P_i in state s accesses a register containing v then the new processor state is $\delta_i(s, v)$.

The computation starts with input x_i stored in register I_i, a designated initial symbol $0 \in V$ in each of the remaining registers, and each processor P_i in a designated initial state $s_i \in S$.

Let $\vec{x} = \langle x_1, \cdots, x_n\rangle$ be the tuple of inputs to the computation. We denote by $s_i^t(\vec{x})$ the state of processor P_i, and by $c_j^t(\vec{x})$ the content of register R_j, at step t of the computation with input $\vec{x}$. The dependency on $\vec{x}$ will be omitted when it is obvious from the context.

We have

$$\begin{aligned}
&s_i^0 = s_i;\\
&c_j^0 = x_i \text{ if } R_j = I_i,\\
&\quad = 0 \text{ otherwise};\\
&s_i^{t+1} = \delta_i(s_i^t, c_j^t) \text{ if } \alpha_i(s_i^t) = \langle j, R\rangle \text{ or } \alpha_i(s_i^t) = \langle j, RW\rangle;\\
&c_j^{t+1} = \omega_i(s_i^t) \text{ if } \alpha_i(s_i^t) = \langle j, RW\rangle\\
&\quad = c_j^t \text{ otherwise.}
\end{aligned}$$

We require in the EREW model that the first components of $\alpha_i(s_i^t)$ and $\alpha_j(s_j^t)$ be distinct for any $i \neq j$. In the CREW model we require that if $\alpha_i(s_i^t) = \langle k, RW\rangle$, then the first component of $\alpha_j(s_j^t)$ is distinct from k for any $j \neq i$. In both cases the processor states and register contents are well defined.

The outputs of the computation are contained at step T in the k output registers. Thus, a paracomputer Π computes the function $F_\Pi : X^n \to X^k$ defined by

$$F_\Pi(x_1, \cdots, x_n) = c_{j_1}^T(\vec{x}), \cdots, c_{j_k}^T(\vec{x}),$$

where $R_{j_1}, \cdots, R_{j_k}$ are the k output registers.

Note that we do not restrict the size of the alphabet, or the number of processor states (they may both be infinite).

A function F is *finite* if it has finite range, say $\{0, \cdots, k\}$. Decision problems are represented by finite functions. For example, the range search problem is associated with the function $RS(x_1, \cdots, x_n, y) = \max\{i: x_i < y\}$, defined on all $(n+1)$-tuples of distinct elements that fulfill the condition $x_1 < \cdots < x_n$. The discrete root finding problem is associated with the function $DRF(x_1, \cdots, x_n) = \max(i: x_i = 0\}$, defined on all n-tuples of elements from the set $\{0, 1\}$ that fulfill the condition $x_1 \leqq \cdots \leqq x_n$.

Let $F: X^n \to \{0, \cdot\cdot, k\}$ be a finite function. A paracomputer *computes a unary encoding* of F if it has n input registers and $k+1$ output registers, and at the end of the computation with input $\vec{x}$ the only output register that has been modified is the register with index $F(\vec{x})$. If a paracomputer with time bound T computes the finite function F, then a unary encoding of F can be computed by a paracomputer with time bound $T+1$ and the same number of processors. It will be more convenient to prove lower bounds for unary computations, since we have to consider only the indices of the registers accessed, but not the values stored.

Let Π and Π' be paracomputers with the same number of processors and registers, and the same set of input symbols. Then Π and Π' are *access equivalent* if for each input $\vec{x}$, each i, and each t, $\alpha_i(s_i^t(\vec{x})) = \alpha_i'(s_i'^t(\vec{x}))$. If a unary encoding is used, then two access equivalent paracomputers compute the same function.

Let us introduce the following definitions. The set $[a, b] = \{a, a+1, \cdots, b\}$ is called a *segment*; we denote by $\bar{n}$ the segment $[0, n]$. Let $\mathcal{S} = \{S_1, \cdots, S_k\}$ be a family of subsets of $\bar{n}$. We say that r is a *critical point* of $\mathcal{S}$ if there exist a set S_i such that $r-1 \in S_i \Leftrightarrow r \notin S_i$. Let $0 < r_1 < \cdots < r_s$ be the critical points of the family $\mathcal{S}$. The *segment partition defined by* $\mathcal{S}$ consists of the segments $[0, r_1 - 1], [r_1, r_2 - 1], \cdots, [r_s, n]$. The segment partition defined by $\mathcal{S}$ is the coarsest partition of $\bar{n}$ into segments that refines the partition defined by the 2^k sets $\bigcap_i^k S_i^{\varepsilon_i}$, where $\varepsilon_i \in \{0, 1\}$, $S^0 = S$, and $S^1 = \bar{n} - S$. In particular, if $\mathcal{S}$ is a partition of $\bar{n}$, then the segment partition defined by $\mathcal{S}$ is the coarsest partition of $\bar{n}$ into segments that refines the partition $\mathcal{S}$.

THEOREM 3.1. *$\Omega(\lg n - \lg p)$ steps are required to solve the discrete root finding problem for a sequence of length n on an EREW machine with p processors.*

Proof. The lower bound results from the lack of a mechanism to distribute instantaneously information throughout the system. In order to prove it we have to trace at each step the "information" represented by the state of each processor and the content of each register in shared memory. We do that at each step for all the possible input values, thus obtaining a "synoptic" description of the possible computations.

The information represented by the state of a processor (the content of a register) consists of the set of input values that could produce this state (content). It is important to note that the information represented by the content of a register may change even if this register is not modified, as the fact that no processor stored a new value is informative in itself. Cook and Dwork show in [4] how such "negative" information can be used at profit.

With each processor (register) is associated at each step a partition of the input symbols, according to the distinct states (values) the processor (register) may assume at this step. The lower bound will be obtained by tracing the evolution of these partitions, and showing that the number of sets in these partitions cannot grow too fast. In fact, we shall not trace the partitions that obtain in an actual computation, but the partitions that would obtain in a computation where there is no "loss of information", i.e. a computation where a processor stores a complete account of the information it has whenever it writes in shared memory. These partitions depend only on the access pattern of the processors.

Rather than counting the number of classes in each partition, it is easier to count the number of critical points of the partition.

Let Π be an EREW paracomputer with p processors, q registers, and time bound T, that computes a unary encoding of the finite function DRF associated with the discrete root finding problem for sequences of length n. We define inductively sets $P(i, t)$, $i = 1, \cdots, p$, and $R(j, t)$, $j = 1, \cdots, q$, such that $P(i, t)(R(j, t))$ contains all the critical points of the partition defined by processor P_i (register R_j) at step t of the computation.

(i) $P(i, 0) = \phi$, $i = 1, \cdots, p$.
(ii) $R(j, 0) = \{i\}$ if R_j initially contains the ith input
$= \phi$ otherwise.
(iii) $r \in P(i, t)$ iff
(a) $r \in P(i, t-1)$, or
(b) P_i accesses at step t of the computation on input $\vec{w}_r$ the register R_j, and $r \in R(j, t-1)$.
(iv) $r \in R(j, t)$ iff
(a) $r \in R(j, t-1)$, or
(b) P_i modifies at step t of the computation with input $\vec{w}_r$ the register R_j, and $r \in P(i, t-1)$, or
(c) P_i modifies at step t of the computation with input $\vec{w}_{r-1}$ the register R_j, and $r \in P(, t-1)$.

These definitions are illustrated in Fig. 1.

CLAIM.
(i) If $r \notin P(i, t)$, then $s_i^t(\vec{w}_r) = s_i^t(\vec{w}_{r-1})$.
(ii) If $r \notin R(j, t)$, then $c_j^t(\vec{w}_r) = c_j^t(\vec{w}_{r-1})$.

Proof. By induction on t. The claim is obvious for $t = 0$. We suppose it holds for $t-1$ and prove it for t.

(i) Since $r \notin P(i, t-1)$, $s_i^{t-1}(\vec{w}_r) = s_i^{t-1}(\vec{w}_{r-1})$. In particular, $\alpha_i(s_i^{t-1}(\vec{w}_r)) = \alpha_i(s_i^{t-1}(\vec{w}_{r-1}))$. Let j be the index of the register accessed. If $r \notin R(j, t-1)$ then, by the

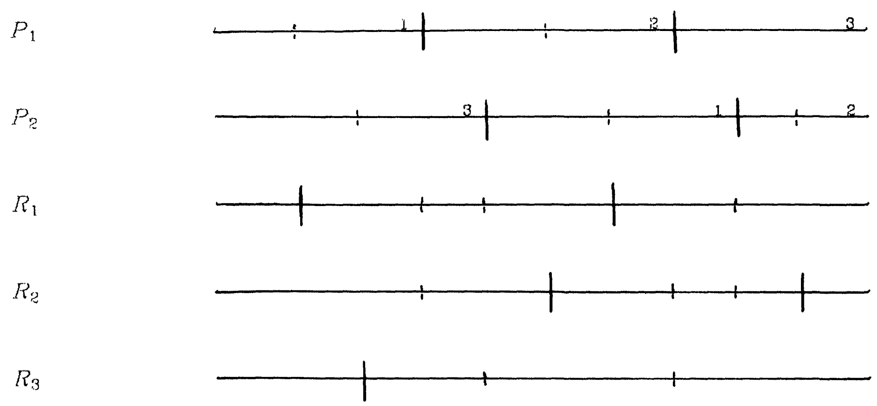

FIG. 1. *Evolution of partitions over one step.* | *Marks critical points at step* $t-1$; ¦ *marks new critical points at step* t.

inductive assertion, $c_j^{t-1}(\vec{w}_r)=c_j^{t-1}(\vec{w}_{r-1})$, so that

$$s_i^t(\vec{w}_r)=\delta_i(s_i^{t-1}(\vec{w}_r),\, c_j^{t-1}(\vec{w}_r))=\delta_i(s_i^{t-1}(\vec{w}_{r-1}),\, c_j^{t-1}(\vec{w}_{r-1}))=s_i^t(\vec{w}_{r-1}).$$

On the other hand, if $r\in R(j, t-1)$, then $r\in P(i, t)$.

(ii) As $r\notin R(j, t-1)$, $c_j^{t-1}(\vec{w}_r)=c_j^{t-1}(\vec{w}_{r-1})$. Suppose that P_i modifies R_j at step t of the computation with input $\vec{w}_r$. If $r\notin P(i, t-1)$, then, by the inductive assumption, $s_i^{t-1}(\vec{w}_r)=s_i^{t-1}(\vec{w}_{r-1})$, so that

$$c_j^t(\vec{w}_r)=\omega_i(s_i^{t-1}(\vec{w}_r))=\omega_i(s_i^{t-1}(\vec{w}_{r-1}))=c_j^t(\vec{w}_{r-1}).$$

On the other hand, if $r\in P(i, t-1)$ then $r\in R(j, t)$. A similar argument applies if R_j is modified in the computation with input $\vec{w}_{r-1}$. If R_j is not modified at step t neither in the computation with input $\vec{w}_r$ nor in the computation with input $\vec{w}_{r-1}$ then

$$c_j^t(\vec{w}_r)=c_j^{t-1}(\vec{w}_r)=c_j^{t-1}(\vec{w}_{r-1})=c_j^t(\vec{w}_{r-1}).$$

Let $c(t)$ be an upper bound on the number of critical points.

$$c(t)=\sum_i |P(i, t)|+\sum_j \max(0, |R(j, t)|-1).$$

Initially

$$c(0)=0. \tag{3.1}$$

If $R_j=O_i$ then R_j is not modified during the computations with input $\vec{w}_{i-1}$ or $\vec{w}_{i+1}$, and is modified during the computation with input $\vec{w}_i$. It follows that $i, i+1\in R(j, T)$, and

$$c(T)\geqq n. \tag{3.2}$$

We shall end our proof by showing that

$$c(t)\leqq 4c(t-1)+p. \tag{3.3}$$

Indeed, (3.1) and (3.3) imply that

$$c(t)\leqq\frac{4^t-1}{3}p,$$

which together with (3.2) yields the inequality

$$n\leqq\frac{4^T-1}{3}p, \quad\text{or}\quad T\geqq\log_4\left(3\frac{n}{p}+1\right)\geqq\frac{1}{2}(\lg n-\lg p).$$

Each occurrence of r in a set $P(i, t-1)$ contributes at most one new occurrence of r in a set $R(j, t)$ according to rule (iv.b) and one new occurrence according to rule (iv.c). Thus

$$\sum_j |R(j, t)| \leqq \sum_j |R(j, t-1)| + 2\sum_i |P(i, t-1)|.$$

Since $|R(j, t)| \geqq R|(j, t-1)|$ this implies that

$$\sum_j \max(0, |R(j, t)|-1) \leqq \sum_j \max(0, |R(j, t-1)|-1) + 2\sum_i |P(i, t-1)|.$$

At most one processor may access at step t of a computation on input $\vec{w}_r$ the register R_j (this is the point where we are using the EREW property). Thus each occurrence of r in a set $R(j, t-1)$ contributes at most one new occurrence of r in a set $P(i, t)$ according to rule (iii.b). Let J_t be the set of registers accessed by some processor at step t of some computation. The number of distinct registers accessed by P_i at step t of some computation is bounded by the number of segments in the segment partition determined by the points in the set $P(i, t-1)$, i.e. by $|P(i, t-1)|+1$. It follows that $|J_t| \leqq \sum_i^p |P(i, t-1)| + p$.

We obtain the inequality

$$\begin{aligned}\sum_{i=1}^{p} |P(i, t)| &\leqq \sum_{i=1}^{p} |P(i, t-1)| + \sum_{j\in J_t} |R(j, t-1)| \\ &= \sum_i |P(i, t-1)| + \sum_{j\in J_t} (|R(j, t-1)|-1) + |J_t| \\ &\leqq 2\sum_i |P(i, t-1) + \sum_j \max(0, |R(j, t-1)|-1) + p.\end{aligned}$$

Combining the last two inequalities one obtains that

$$\begin{aligned}c(t) &= \sum_i |P(i, t)| + \sum_j \max(0, |C(j, t)|-1) \\ &\leqq 4\sum_i |P(i, t-1)| + 2\sum_j \max(0, |C(j, t-1)|-1) + p \\ &\leqq 4c(t-1) + p.\end{aligned}$$

□

We did not use in the proof the fact that concurrent writes are not allowed. Indeed, the lower bound is still valid for an ERCW (exclusive read, concurrent write) parallel machine. It is also valid even if inputs are initially replicated, so that each input value can be accessed concurrently by all the processors.

The last lower bound is optimal. An EREW machine with $n+1$ processors can solve the root finding problem for n keys $x_1, \cdots, x_n$ in constant time by comparing in parallel x_i to x_{i+1}, $i=0, \cdots, n$ ($x_0=0$, $x_{n+1}=1$, by definition). This generalizes to an algorithm that solves the problem with p processors $P_1, \cdots, P_p$ in $O(\lg(n/p))$ steps as follows. Let $i_j = \lfloor j(n+1)/p \rfloor$. Firstly, each processor P_j checks in parallel whether $x_{i_{j-1}} < x_{i_j}$. Next, the unique processor P_j that found a strong inequality continues to execute a serial bisection algorithm on the list $x_{i_{j-1}+1}, \cdots, x_{i_j-1}$.

This simple algorithm can be extended to solve by comparisons the general range searching problem in $O(\lg n - \lg p)$ steps, provided that the searched key can be accessed concurrently by all the processors.

4. Searching with an unbounded number of processors.

If the searched key cannot be accessed concurrently by all the processors, then $\Omega(\lg p)$ steps are required to distribute it, thus cancelling the gain obtained from parallelism in the last algorithm. This would seem to imply that $\Omega(\lg n)$ steps are required to solve the range search problem, independently of the number of processors. It turns out, however, that the range search problem can be solved much faster.

THEOREM 4.1. *The range search problem for a table of size n can be solved by an EREW paracomputer with $O(n)$ processors and $O(n)$ registers in $O(\sqrt{\lg n})$ steps.*

Proof. The idea of the algorithm is illustrated in Fig. 2. The search is carried according to a multiway search tree where the branching factor is doubled at each level (Fig. 2b). Such a tree of depth t contains $2^{t(t-1)/2}-1$ keys, so that searching a table of that size requires t accesses to nodes of the tree. An access to a node of this tree is done in one memory access provided that an encoding of the tuple of keys at that node has been stored in one memory cell. Once the encoding of the keys at the node has been read, the decoding and the subsequent comparisons are performed locally, i.e. at no cost.

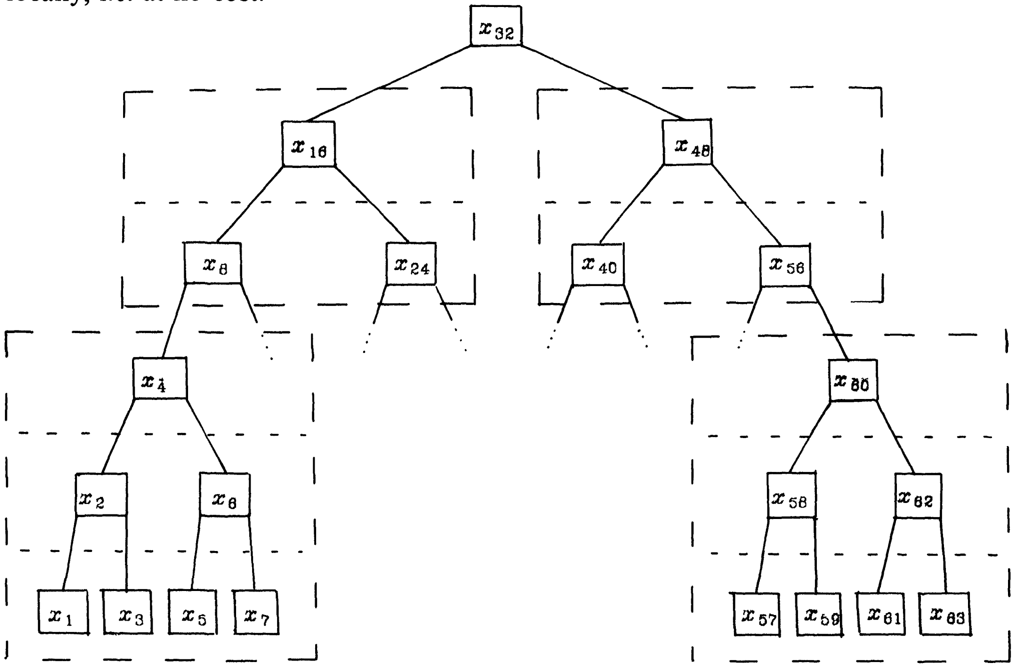

FIG. 2a. *Binary search trace and its compression.*

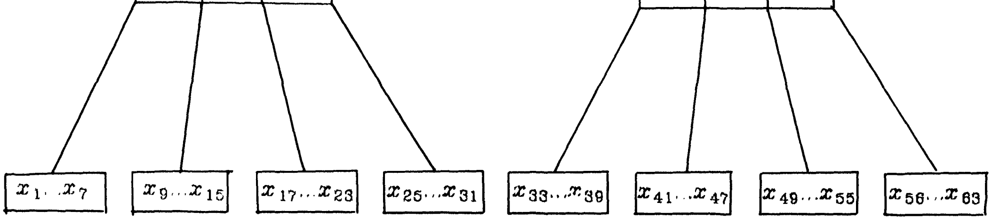

FIG. 2b. *Corresponding multiway search tree.*

The search will be carried by one processor. Concurrently, the remaining processors will compute and store encodings of tuples of keys. The multiway search tree is obtained by "compressing" the binary search tree: A node at level i of the multiway search tree contains the keys belonging to a complete subtree of depth i, rooted at level $\frac{1}{2}i(i-1)+1$, in the binary tree (Fig. 2a). The processors will compress the binary search tree, increasing by one at each iteration the depth of the subtrees which encodings has been computed.

We describe now this algorithm more formally. Assume w.l.g. that $n+1=2^{t(t+1)/2}$. Let $a_k=\sum_{i=1}^{k} i=\frac{1}{2}k(k+1)$. The algorithm consists of t iterations. At the end of iteration i the search has proceeded through i levels of the multiway search tree, that correspond to a_i levels of the binary search tree. Also, encodings have been computed for the keys belonging to each complete subtree of the binary tree that has depth $i+1$ and has its leaves at level a_j, $j=i+1,\cdots,t$ (see Fig. 2a). In particular, encodings have been computed for the keys of each subtree of depth $i+1$ rooted at level a_i+1, i.e. for each tuple of keys belonging to a node of the multiway search tree at level $i+1$. During iteration $i+1$ the searching processor accesses one of these encodings to push the search one level down on the multiway search tree; each of the remaining processors computes an encoding of the keys belonging to a binary tree of depth $i+2$ by combining the encodings for the left and right subtrees which have been computed at the previous iteration, and the key at the root.

At each iteration, each key and each encoding is accessed at most once, and the total number of new encodings computed is less than n. It follows that each iteration can be performed in constant time using $O(n)$ processors, and that only $O(n)$ registers are needed. The total running time is $O(t)=O(\sqrt{\lg n})$. □

5. Order invariant computations and canonical paracomputers. We prove in this section that algorithms using only comparisons are as powerful as more general algorithms in solving comparison based problems. In order to do so, it will be convenient to work with a paracomputer model where information of input values is clearly distinct from other information.

Let $X^{\leqq n}$ be the set of strings over X of length at most n. A paracomputer Π with n inputs is in *canonical form* if it fulfills the following conditions.

(i) The set of processor states is of the form $X^{\leqq n}\times C$ and the set of register symbols is of the form $X^{\leqq n}\times D$ (n is the number of inputs to Π). (The input symbol $x\in X$ is identified with the register symbol $\langle x, 0\rangle$, where $0\in D$ is a fixed constant.)

(ii) If $\omega_i(\langle\sigma, s\rangle)=\langle\tau, v\rangle$, then every element of τ occurs in σ (a processor can write an input symbol only if it is present in its "local memory").

(iii) If $\delta_i(\langle\sigma, s\rangle, \langle\tau, v\rangle)=\langle\sigma', s'\rangle$ then every element of σ' occurs either in σ or in τ (a processor can store an input symbol in its "local memory" only if it is already there, or if it accessed it from shared memory).

We call C the set of *control symbols*, and D the set of *coordination symbols.*

The behaviour of a canonical paracomputer is conveniently indicated by specifying the indices of the input symbols that are moved. The functions $\bar{\omega}_i$ and $\bar{\delta}_i$ are defined by the following identities.

$$\bar{\omega}_i(\langle\sigma_1\cdots\sigma_k, c\rangle)=\langle j_1\cdots j_r, d\rangle, \quad \text{where}$$

$$\omega_i(\langle\sigma, c\rangle)=\langle\sigma_{j_1}\cdots\sigma_{j_r}, d\rangle.$$

$$\bar{\delta}_i(\langle\sigma_1\cdots\sigma_k, c\rangle, \langle\sigma_{k+1}\cdots\sigma_r, d\rangle)=\langle j_1\cdots j_s, e\rangle, \quad \text{where}$$

$$\delta_i(\langle\sigma_1\cdots\sigma_k, c\rangle, \langle\sigma_{k+1}\cdots\sigma_r, d\rangle)=\langle\sigma_{j_1}\cdots\sigma_{j_s}, e\rangle.$$

The definition of a canonical paracomputer is motivated by the following result.

THEOREM 5.1. *To each paracomputer* Π *with* p *processors,* q *registers, and time bound* T*, we can associate an access equivalent canonical paracomputer* $\mathcal{F}(\Pi)$ *with* $O(2^T(p+q))$ *control and coordination symbols.*

We postpone the proof of this theorem to the next section. It is important to note that the number of control and coordination symbols of $\mathcal{F}(\Pi)$ is independent of the number of input symbols.

The following definitions will make precise the notion of a comparison based computation. Two strings $\vec{x}, \vec{y} \in X^n$ are *order equivalent* ($\vec{x} \equiv \vec{y}$) if $\forall i, j,\ x_i < x_j \Leftrightarrow y_i < y_j$. A function F defined on X^n is *order invariant* if $\vec{x} \equiv \vec{y} \Rightarrow F(\vec{x}) = F(\vec{y})$. F is order invariant iff each set $F^{-1}(y)$ can be defined by inequalities, that is by Boolean combinations of assertions of the form $x_i < x_j$. The range search function RS is order invariant. Note that an order invariant function has finite range.

A canonical paracomputer is *order invariant* if, when $\sigma \equiv \sigma'$ and $\tau \equiv \tau'$, the following conditions are fulfilled for any c, d and i.

(i) $\alpha_i(\langle \sigma, c\rangle) = \alpha_i(\langle \sigma', c\rangle)$.

(ii) $\bar{\omega}_i(\langle \sigma, c\rangle) = \bar{\omega}_i(\langle \sigma', c\rangle)$.

(iii) $\bar{\delta}_i(\langle \sigma, c\rangle, \langle \tau, d\rangle) = \bar{\delta}_i(\langle \sigma', c\rangle, \langle \tau', d\rangle)$.

Informally, a canonical paracomputer is order invariant if the behavior of each processor depends only on the value of the control and coordination symbols, and the relative order of the input values it has access to, but not on the value of the input symbols themselves. In particular, the computation will follow the same course on two sets of input values where the inputs have the same relative order.

Let $\sigma|_{\vec{y}}^{\vec{x}}$ denote the string obtained by substituting in σ each occurrence of x_i by an occurrence of y_i. We leave to the reader the straightforward proof of the following lemma, which formalizes the last claim.

LEMMA 5.2. *Let* Π *be an order invariant canonical paracomputer, and let* $\vec{x}$ *and* $\vec{y}$ *be order equivalent input vectors. Then the following holds*

(i) *If* $s_i^t(\vec{x}) = \langle \sigma, c\rangle$ *and* $s_i^t(\vec{y}) = \langle \sigma', c'\rangle$ *then* $c = c'$ *and* $\sigma' = \sigma|_{\vec{y}}^{\vec{x}}$.

(ii) *If* $c_j^t(\vec{x}) = \langle \sigma, d\rangle$ *and* $c_j^t(\vec{y}) = \langle \sigma', d'\rangle$ *then* $d = d'$ *and* $\sigma' = \sigma|_{\vec{y}}^{\vec{x}}$.

(iii) $\alpha_i(s_i^t(\vec{x})) = \alpha_i(s_i^t(\vec{y}))$ *for all* i *and* t.

COROLLARY 5.3. *Let* F *be an order invariant function defined on* $U \subset X$*, and let* $W \subset X^n$ *be a set that contains a representative for each order equivalence class in* U (i.e. $\forall \vec{x} \in U\ \exists \vec{y} \in W$ s.t. $\vec{x} \equiv \vec{y}$). *Let* Π *be an order invariant canonical paracomputer that computes a unary encoding of* F *for inputs taken from* W*. Then* Π *computes a unary encoding of* F *for any input taken from* U.

Proof. Let $\vec{x}$ be an input vector from U. Let $\vec{y} \in W$ be order equivalent to $\vec{x}$. Then $F(\vec{x}) = F(\vec{y})$. On the other hand, by Lemma 5.2, the computation of Π on $\vec{x}$ is access equivalent to the computation on $\vec{y}$, so that a unary encoding of $F(\vec{x})$ is computed. □

We make use of the following well-known theorem [9].

RAMSEY'S THEOREM. *For any* k, m *and* t *there exist a number* $N(k, m, t)$ *such that the following is true: Let* S *be a set of size at least* $N(k, m, t)$*; if we divide the* k*-element subsets of* S *into* t *parts, then at least one part contains all the* k*-element subsets of some* m *elements of* S.

THEOREM 5.4. *For each* m, p, q *and* T *there exist a number* $N = N(m, p, q, T)$ *such that the following holds: Let* Π *be a canonical paracomputer with* p *processors,* q *registers, time bound* T*, and an input set* X *of size* $|X| \geqq N$*. Then there exists a subset* $Y \subset X$ *such that* $|Y| \geqq m$ *and* Π *is order invariant when restricted to inputs from* Y.

Proof. Let $\{x_1 \cdots x_n\}$ and $\{y_1 \cdots y_n\}$ be two n-element sets of input symbols, indexed in increasing order. We say that $x_1 \cdots x_n$ *is congruent to* $y_1 \cdots y_n$ if the following holds:

If σ and τ are strings from $X^{\leqq n}$ with symbols taken from the set $\{x_1 \cdots x_n\}$, $\sigma' = \sigma|_{\bar{y}}^{\bar{x}}$, and $\tau' = \tau|_{\bar{y}}^{\bar{x}}$, then $\alpha_i(\langle\sigma, c\rangle) = \alpha_i(\langle\sigma', c\rangle)$, $\bar{\omega}_i(\langle\sigma, c\rangle) = \bar{\omega}_i(\langle\sigma', c\rangle)$, and $\bar{\delta}_i(\langle\sigma, c\rangle, \langle\tau, d\rangle) = \bar{\delta}_i(\langle\sigma', c\rangle, \langle\tau', d\rangle)$, for any c, d and i.

It is easy to see that this is indeed an equivalence relation on the n-element subsets of X. The number of distinct values the functions α_i, $\bar{\omega}_i$, and $\bar{\delta}_i$ may assume is bounded by a function of n, q, $|C|$ and $|D|$. It follows that the number G of distinct congruence classes is bounded by a function of n, p, q, $|C|$ and $|D|$. According to Ramsey's theorem, for any s there is a number $N = N(n, G, s)$ such that if $|X| \geqq N$ then X contains a subset Y such that $|Y| \geqq s$ and all n-element subsets of Y belong to the same congruence class. This entails that, if σ, σ', τ, and τ' are members of $Y^{\leqq n}$, $\sigma \equiv \sigma'$ and $\tau \equiv \tau'$ then $\alpha_i(\langle\sigma, c\rangle) = \alpha_i(\langle\sigma', c\rangle)$, $\bar{\omega}_i(\langle\sigma, c\rangle) = \bar{\omega}_i(\langle\sigma', c\rangle)$, and $\bar{\delta}_i(\langle\sigma, c\rangle, \langle\tau, d\rangle) = \bar{\delta}_i(\langle\sigma', c\rangle, \langle\tau', d\rangle)$, for any i, c and d. □

COROLLARY 5.5. *For each p, q and T there exist a number $N = N(p, q, T)$ such that the following holds: Let F be a finite order invariant function defined on X. Let Π be a canonical paracomputer with p processors, q registers and time bound T, that computes a unary encoding of F. Let $|X| \geqq N$. Then there exists an order invariant canonical paracomputer with the same number of processors and registers, and the same time bound that computes F.*

Proof. Let n be the number of variables of F ($n \leqq q$). According to Theorem 5.4, if X is large enough, then there exists a set Y such that $|Y| \geqq n$ and Π is order invariant when restricted to inputs from Y. Each string from $X^{\leqq n}$ is order equivalent to a string from $Y^{\leqq n}$.

Let Π' be the canonical paracomputer defined as follows: Π' has the same number of processors and registers, same sets of symbols, and the same time bound as Π; the access, write and transition functions are defined as follows.

$$\alpha_i'(\langle\sigma, c\rangle) = \alpha_i(\langle\sigma', c\rangle)$$

where $\sigma' \in Y^{\leqq n}$ is order equivalent to σ;

$$\bar{\omega}_i'(\langle\sigma, c\rangle) = \bar{\omega}_i(\langle\sigma', c\rangle)$$

where $\sigma' \in Y^{\leqq n}$ is order equivalent to σ;

$$\bar{\delta}_i'(\langle\sigma, c\rangle, \langle\tau', d\rangle) = \bar{\delta}_i(\langle\sigma', c\rangle, \langle\tau', d\rangle)$$

where $\sigma' \in Y^{\leqq n}$ is order equivalent to σ and $\tau' \in Y^{\leqq n}$ is order equivalent to τ.

As Π is order invariant on inputs from Y^n, Π' is well defined, and order invariant on all inputs from X^n. Also, the computations of Π' are identical to the computations of Π for inputs taken from Y^n. Thus Π' computes a unary encoding of F on Y^n, and by Corollary 5.3, computes a unary encoding of F on all X^n. □

6. Lower bounds on searching with an unbounded number of processors. We prove in this section that $\Omega(\sqrt{\lg n})$ lower bound on searching with an unbounded number of processors. The argument consists of three parts. Firstly, we shall complete the proof of Theorem 5.1, thereby reducing the problem to canonical paracomputers. Secondly, the results of the previous section can be used to reduce the problem to canonical order invariant paracomputers. Finally, an argument similar to that used in the proof of Thm. 3.1 yields the lower bound.

Proof of Theorem 5.1. We shall build $\mathcal{F}(\Pi)$ from Π by stipulating that whenever a processor of Π writes onto shared memory, then the corresponding processor of

$\mathscr{F}(\Pi)$ writes onto shared memory a complete account of the information available to it; whenever a processor of Π reads from shared memory, the corresponding processor of $\mathscr{F}(\Pi)$ reads and stores in its local memory (its state) the content of the register accessed. Thus, each processor of $\mathscr{F}(\Pi)$ has at each step sufficient information to simulate the corresponding processor of Π.

In a processor state $\langle\sigma, c\rangle$, σ will contain the input symbols which values "are known" to the processor, and c will represent the knowledge of the processor on the memory accesses that were executed. A similar convention holds for register values.

We shall use *S-expressions* to encode information on memory accesses. $L \in \mathscr{S}(X)$, the set of S-expressions over the set X, iff

$L = NIL$, or

$L \in X$, (L is an *atom* from C) or

$L = (L_1 \cdot L_2)$,

where L_1 and L_2 are S-expressions over X. The list $(L_1 \cdots L_k)$ is the S-expression $(\cdots (L_1 \cdot L_2) \cdots) \cdot L_k)$ (this is the reverse of LISP convention).

Let Π be a paracomputer. $\mathscr{H}(P_i, \vec{x}, t)$, the *history of processor P_i at step t of the computation on input $\vec{x}$*, and $\mathscr{H}(R_j, \vec{x}, t)$, the *history of register R_j at step t of the computation on input $\vec{x}$*, are lists defined inductively as follows.

$$\mathscr{H}(P_i, \vec{x}, 0) = \hat{i},$$

where $\hat{i}$ is an S-expression with no atoms, encoding the number i;

$$\mathscr{H}(R_j, \vec{x}, 0) = x_i \quad \text{if } R_j \text{ contains the } i\text{th input}$$
$$= NIL, \quad \text{otherwise.}$$

If processor P_i accesses register R_j at step t of the computation with input $\vec{x}$ then

$$\mathscr{H}(P_i, \vec{x}, t) = (\mathscr{H}(P_i, \vec{x}, t-1)(\mathscr{H}(R_j, \vec{x}, t-1)).$$

If processor P_i modifies register R_j at step t of the computation with input $\vec{x}$, then

$$\mathscr{H}(R_j, \vec{x}, t) = \mathscr{H}(P_i, \vec{x}, t-1);$$

otherwise

$$\mathscr{H}(R_j, \vec{x}, t) = \mathscr{H}(R_j, \vec{x}, t-1).$$

CLAIM. (i) The value of i and the state $s_i^t(\vec{x})$ of P_i at step t of the computation with input $\vec{x}$ are uniquely determined by $\mathscr{H}(P_i, \vec{x}, t)$.

(ii) The content $c_j^t(\vec{x})$ of R_j at step t of the computation with input $\vec{x}$ is uniquely determined by j and $\mathscr{H}(R_j, \vec{x}, t)$.

Proof. The claim is trivially true for $t = 0$. Assume it is valid for $t-1$.

(i) Let $L = \mathscr{H}(P_i, \vec{x}, t)$. The value of i can be determined from the first element of the list L. From L we can extract $\mathscr{H}(P_i, \vec{x}, t-1)$ and determine $s_i^{t-1}(\vec{x})$ and, therefore, the index j of the register accessed by P_i at step t. The history $\mathscr{H}(R_j, \vec{x}, t-1)$ of register R_j at time $t-1$ occurs in L, so that the content $c_j^{t-1}(\vec{x})$ of R_j at time $t-1$ can be determined. But $s_i^{t-1}(\vec{x})$ and $c_j^{t-1}(\vec{x})$ determine $s_i^t(\vec{x})$.

(ii) Let $L = \mathscr{H}(R_j, \vec{x}, t)$. If $L = \text{NIL}$ or $L = x_j$, then no processor wrote on R_j and its content is known. Otherwise $L = \mathscr{H}(P_i, \vec{x}, t'-1)$ where t' is the last step where a processor modified R_j and P_i is the processor that modified R_j at step t'. It is possible to determine from the expression the values of i, and the state $s_i^{t'-1}$ of P_i at step $t'-1$. These determine the next value $c_j^{t'}$ of R_j, which is also the value of R_j at step t.

We define Π' to be a paracomputer with the same number of processors, and registers, the same time bound, and the same set of input symbols as Π. The processor states and register values of Π' are S-expression with atoms taken from X. The functions of Π' are defined as follows.

$$\alpha_i'(L) = \alpha_i(s) \quad \text{if } s = s_i^t(\vec{x}),\ L = \mathscr{H}(P_i, t, \vec{x});$$

$\alpha_i'(L)$ is undefined otherwise.

$$\omega_i'(L) = L.$$

$$\delta_i'(L_1, L_2) = (L_1 \cdot L_2).$$

The previous claim implies that Π' is access equivalent to Π; in fact $s_i'^t(\vec{x}) = \mathscr{H}(P_i, t, \vec{x})$ and $c_j'^t(\vec{x}) = \mathscr{H}(R_j, t, \vec{x})$ for every i, j, t, and $\vec{x}$.

Each history expression of Π contains at most n distinct input symbols, and has length at most $O(2^T(p+q))$. Let L be an S-expression with atoms $x_1, \cdots, x_k$. We represent L by the pair

$$\bar{L} = \langle x_1 \cdots x_k, L|_{\tilde{i}_1 \cdots \tilde{i}_k}^{x_1 \cdots x_k}\rangle,$$

where $\tilde{i}$ is an atom-free S-expression (distinct from $\hat{i}$) that encodes the number i. The canonical paracomputer $\mathscr{F}(\Pi)$ is obtained from Π' by replacing each state symbol, and each register content symbol by its above representation. □

We leave to the reader the proof of the following technical lemma.

LEMMA 6.1. *Let $\mathscr{P}_1, \cdots, \mathscr{P}_r$ be segment partitions of $\bar{n}$. Let $\mathscr{Q}$ be a family of (not necessarily distinct) segments from $\mathscr{P}_1, \cdots, \mathscr{P}_r$ with the property that each element in $\bar{n}$ is contained in at most s sets from $\mathscr{Q}$. Then*

$$|\mathscr{Q}| \leqq \sum_{i=1}^{r} |\mathscr{P}_i| - r + s.$$

THEOREM 6.2. *For any p and q there is a number $N(p, q)$ such that the following holds: If an EREW paracomputer with p processors and q registers solves in time T the range search problem for n inputs taken from a totally ordered set X such that $|X| \geqq N$, then $T \geqq \sqrt{\lg n} + O(1)$.*

Proof. Let Π be an EREW paracomputer that computes in T steps a unary encoding of the function $RS: X^n \to \bar{n}$ that is associated with the range search problem. We can assume w.l.g., by Corollary 5.5, that Π is a canonical, order invariant paracomputer. Let us pick $2n+1$ elements from X, $b_0 < a_1 < b_1 < \cdots a_n < b_n$, and consider the behavior of the algorithm on the $n+1$ sets of inputs $\vec{z}_i = \langle a_1, \cdots, a_n, b_i \rangle i = 0, \cdots, n$. Note that $RS(\vec{z}_i) = i$.

We follow now the same approach as in the proof of Theorem 3.1. Let $s_i^t(\vec{z}_{r-1}) = \langle \sigma, c \rangle$ and $s_i^t(\vec{z}_r) = \langle \sigma', c' \rangle$. The only inputs whose relative order in $\vec{z}_{r-1}$ is different from their relative order in $\vec{z}_r$ are y and x_r. Thus, the state of processor P_i distinguishes between input $\vec{z}_{r-1}$ and input $\vec{z}_r$ if the control symbols are distinct ($c \neq c'$), or the set of inputs accessed are distinct ($\sigma' \neq \sigma|_{\vec{z}_r}^{\vec{z}_{r-1}}$), or neither conditions obtain, but both the values of $y(=b_{r-1}$ or $b_r)$ and of $x_r(=a_r)$ are known to P_i (occur in σ and σ'), in which case σ is not order equivalent to σ'. This motivates the following definitions. We define inductively the sets $P(i, t)$ and $R(j, t)$ as follows.

(i) $P(i, 0) = R(j, 0) = \varnothing$.

(ii) $r \in P(i, t)$ iff

(a) $r \in P(i, t-1)$, or

(b) P_i accesses R_j at step t of the computation on input $\vec{z}_r$, and $r \in R(j, t-1)$, or

(c) P_i accesses R_j at step t of the computation on input $\vec{z}_r$, b_r occurs in $s_i^{t-1}(\vec{z}_r)$, and a_r occurs in $c_j^{t-1}(\vec{z}_r)$, or

(d) P_i accesses R_j at step t of the computation on input $\vec{z}_r$, a_r occurs in $s_i^{t-1}(\vec{z}_r)$, and b_r occurs in $c_j^{t-1}(\vec{z}_r)$.

(iii) $r \in R(j, t)$ iff

(a) $r \in R(j, t-1)$, or

(b) P_i modifies R_j at step t of the computation with input $\vec{z}_r$, and $r \in P(i, t-1)$, or

(c) P_i modifies R_j at step t of the computation with input $\vec{z}_{r-1}$, and $r \in P(i, t-1)$.

CLAIM. (i) Let $s_i^t(\vec{z}_{r-1}) = \langle \sigma, c \rangle$ and $s_i^t(\vec{z}_r) = \langle \sigma', c' \rangle$. If $r \notin P(i, t)$, then $c = c'$, $\sigma' = \sigma|_{\vec{b}_r}^{\vec{b}_{r-1}}$, and $\sigma' \equiv \sigma$.

(ii) Let $c_j^t(\vec{z}_{r-1}) = \langle \tau, d \rangle$ and $c_j^t(\vec{z}_r) = \langle \tau', d' \rangle$. If $r \notin R(j, t)$ then $d = d'$, $\tau' = \tau|_{\vec{b}_r}^{\vec{b}_{r-1}}$, and $\tau \equiv \tau$.

The proof of this claim is similar to the proof of the corresponding claim in Theorem 3.1.

Let

$$c(t) = \sum_{i=1}^{p} |P(i, t)| + \sum_{j=1}^{q} |R(j, t)|.$$

We have $c(0) = 0$ and $c(T) \geqq 2n$. Let us consider now the growth of $c(t)$.

We have the following two facts.

Fact 1. Let f_t be the tth element of the Fibonacci sequence ($f_0 = f_1 = 1$). Then the number of distinct input symbols occurring in the content of a register at step t is bounded by f_{t-1}, and the number of distinct input symbols occurring in the state of a processor at step t is bounded by f_t.

Fact 2. Each input symbol may occur in the states of at most 2^{t-1} processors and the contents of at most 2^{t-1} registers at step t of the computation on a fixed vector of inputs.

Let $K = f_{T-1}$ and $H = 2^{T-1}$. Then each processor state and each register content occurring during a computation of Π contains at most K input symbols, and each input symbol occurs in the states of at most H processors and the contents of at most H registers during the computation on input $\vec{z}_r$.

It is easily seen that

$$\sum_j |R(j, t)| \leqq \sum_j |R(j, t-1)| + 2 \sum_i |P(i, t-1)|. \tag{6.1}$$

The number of points contributed to sets $P(i, t)$ according to rules (ii.1) and (ii.b) is bounded by

$$\sum_i |P(i, t-1)| + \sum_j |R(j, t-1)|. \tag{6.2}$$

It remains to assess the contribution of rules (ii.c) and (ii.d).

Let $\mathcal{P}(i, t)$ be the segment partition associated with $P(i, t)$; let $\mathcal{P}_y(i, t)$ be the set of segments in $\mathcal{P}(i, t)$ corresponding to states of P_i where an input symbol $y = b_r$ occurs; let $\mathcal{R}(j, t)$ be the segment partition associated with $R(j, t)$. For each r there are at most H processors that contain $y = b_r$ in their state at step t of the computation on input $\vec{z}_r$. It follows, by Lemma 6.1, that

$$\sum_i |\mathcal{P}_y(i, t)| \leqq \sum_i (|\mathcal{P}(i, t)| - 1) + H = \sum_i |P(i, t)| + H.$$

Replace each segment $S \in \mathcal{P}_y(i, t-1)$ by the segments $\{S \cap R \colon R \in \mathcal{R}(j, t-1)\}$, where j is the index of the register accessed by P_i when in the states corresponding to the segment S. The total number of segments thus obtained is bounded by

$$\sum_i |\mathcal{P}_y(i, t-1)| + \sum_j |R(j, t-1)| \leqq \sum_i |P(i, t-1)| + H + \sum_j |R(j, t-1)|.$$

Each of these segments contributes at most K new critical points, according to rule (ii.c). Thus, the total number of critical points contributed by rule (ii.c) is bounded by

$$K\left(\sum_i |P(i,t)| + \sum_j |R(j,t)| + H\right). \tag{6.3}$$

A similar argument yields the same bound for the number of new critical points contributed by rule (ii.d).

We obtain from (6.2) and (6.3)

$$\sum_i |P(i,t)| \leqq (2K+1)\left[\sum_i |P(i,t-1)| + \sum_j |R(j,t-1)|\right] + 2HK. \tag{6.4}$$

Inequalities (6.1) and (6.4) imply that

$$\begin{aligned} c(t) &= \sum_i |P(i,t)| + \sum_j |R(j,t)| \\ &\leqq (2K+3)\sum_i |P(i,t-1) + (2K+2)\sum_j |R(j,t-1)| + 2HK \\ &\leqq (2K+3)c(t-1) + 2HK. \end{aligned}$$

It follows that

$$2n \leqq c(T) \leqq 2HK\frac{(2K+3)^T - 1}{2K+2} < H(2K+3)^T,$$

so that

$$n \leqq \tfrac{1}{2}H(2K+3)^T. \tag{6.5}$$

Substituting back for K and H, we obtain

$$n \leqq 2^{T^2+O(T)},$$

which implies that $T \geqq \sqrt{\lg n} + O(1)$. □

The last lower bound is valid even if we allow concurrent reads from those input registers that contain the searched table. It is only the access to the searched key that has to be restricted.

7. Paracomputers with bounded bandwidth. The $O(\sqrt{\lg n})$ algorithm relies heavily on the fact that the content of one register may encode the values of an arbitrary number of inputs, so that an arbitrary amount of information can be transferred in one read or write operation, and processed in one instruction cycle. This is not a realistic assumption.

We can restrict this model by restricting the type of operations that can be performed on inputs. This is the approach usually followed in the analysis of comparison based algorithms, where it is assumed that inputs are atomic entities that can be only compared. We obtain a "structured" computational model (in the sense used by [3]), which is more amenable to analysis.

Such restriction runs against the basic approach of this paper which is that of assuming powerful computational nodes, but restricted communication ability. We shall instead impose "structure" on the type of items that can be transmitted in one access to memory. We shall assume that a memory register may contain a unique input symbol; it can also contain a communication symbol, taken from a small set. Inputs are transferred atomically, so that an input symbol cannot encode the values of a tuple of input symbols. Formally, let Π be a paracomputer with input set X and set of register symbols V. Then Π has *memory bandwidth d* if the following two conditions hold:

(i) $V = X \times D$, where D is a set of d communication symbols.

(ii) A processor may write a symbol $x \in X$ only if it had read it at previous steps.

A paracomputer has *bounded memory bandwidth* if it has memory bandwidth d, for some finite d.

We shall allow the memory bandwidth d to grow as a function of the problem size n, but assume it is fixed with respect to the size of the input set.

This restriction is still not sufficient to imply an $\Omega(\lg n)$ lower bound. Indeed, it is still possible to represent the values of a tuple of keys by the state of a processor. We obtain

THEOREM 7.1. *The range searching problem for a table of size n can be solved by an EREW paracomputer with $O(n)$ processors and registers and $O(1)$ memory bandwidth in time $O(\lg n/\lg\lg n)$.*

Proof. The algorithm used is similar to that given in Theorem 4.1. Assume w.l.g. that $n = (t+1)! - 1$. The search proceeds according to a multiway search tree of depth $t = O(\lg n/\lg\lg n)$, where nodes at level i contain i keys, and have, therefore, $i+1$ children (see Fig. 3). Such a tree, of depth t, contains $(t+1)! - 1$ keys, so that a table of that size can be searched in t iterations.

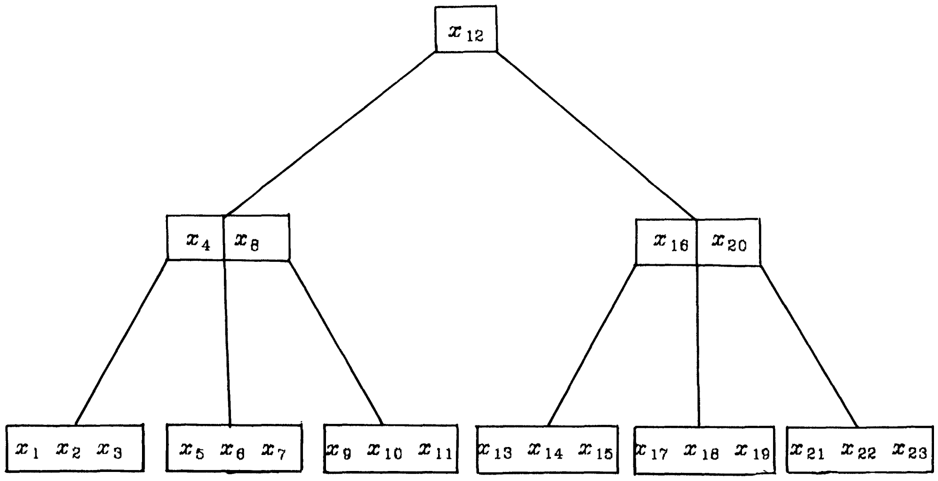

FIG. 3. *Multiway search tree for algorithm in Theorem.* 7.1.

A processor is assigned to each node of that tree. This processor reads at each iteration one key, and stores its value in its local memory. At iteration i the processors assigned to nodes at level i have accessed all the keys at their node. Each processor is also assigned a mailbox. The searched key is initially in the mailbox of the processor assigned to the root. At iteration i the processors assigned to level i nodes access their mailbox. One processor finds the searched key in its mailbox, and compares it to the keys of its node, thereby selecting a node at level $i+1$ where the search proceeds. It then puts the searched key in the mailbox associated with the node selected.

It is easy to see that each iteration can be implemented in constant time, using $O(n)$ processors, $O(n)$ registers, and two communication symbols.

A matching lower bound can be proven, using the methods of the previous section. We leave to the reader the proof of the following analogue to Theorem 5.1.

LEMMA 7.2. *To each paracomputer Π of bounded memory bandwidth, with p processors, q registers, time bound T, and set of communication symbols D we can associate an access equivalent canonical paracomputer $\mathcal{F}(\Pi)$ such that the set of processor states is of the form $X^{\leqq n} \times C$ and the set of register symbols is of the form $X \times D$, with $|C| \leqq O(2^T(p+q+|D|))$.*

THEOREM 7.3. *For any p, q and d there is a number $N = N(p, q, d)$ such that the following holds: If an EREW paracomputer with p processors, q registers, and memory bandwidth d solves in time T the range search problem for n inputs taken from a totally ordered set X such that $|X| \geqq N$, then $T \geqq \lg n/\lg \lg n + O(1)$.*

Proof. The proof is similar to the proof of Theorem 6.2. We can assume w.l.g. that Π is a canonical paracomputer of the form given by Lemma 7.2, and order invariant. Let the $n+1$ input tuples $\vec{z}_0, \cdots, \vec{z}_n$, the sets $P(i, t)$ and $R(j, t)$, and the sequence $c(t)$ be defined as in Theorem 6.2.

We have the following two facts.

Fact 1. At most t input symbols occur in the state of a processor at step t.

Fact 2. A fixed input symbol may occur in at most 2^{t-1} processor states and 2^{t-1} register contents at step t of a computation on a fixed input.

It follows that inequality (6.5) of Theorem 6.2 is valid with $K = T-1$ and $H = 2^{T-1}$. We obtain that

$$n \leqq \tfrac{1}{2}H(2K+3)^T \leqq \tfrac{1}{4}(4T+2)^T$$

so that $T \geqq \lg n/\lg \lg n + O(1)$. □

We further weaken our computational model by restricting the type of information that a processor may store in its local memory (i.e. its "state"). We assume now that each processor has a fixed number of local registers. Each local register, may store an input symbol. In addition, each processor has a finite state control.

Formally, let Π be a paracomputer of bounded memory bandwidth with input set X, set of states S, and set of communication symbols D. Then Π has *processor bandwidth* (k, c), if the following conditions hold:

(i) $S = X^k \times C$, where C is a set of c *control state symbols.*

(ii) Each input symbol that occurs in $\omega_i(\langle \sigma, c \rangle)$ occurs in σ.

(iii) If $\delta_i(\langle \sigma, d \rangle, u) = \langle \sigma', d' \rangle$, then each symbol of σ' occurs either in σ or in u.

The second condition states that the value of a local register at step t is either the value of a local register at step $t-1$ or a value read from memory. The third condition states that a processor may write an input symbol only if it is stored in one of its local registers. Note that a paracomputer of bounded processor bandwidth is a canonical paracomputer (provided that $k \leqq n$).

THEOREM 7.4. *For any p, q, c, d, k, and T there is a number $N = N(p, q, c, d, k, T)$ such that the following holds: If an EREW paracomputer with p processors, q registers, memory bandwidth d, processor bandwidth (c, k), time bound T solves the range search problem for n inputs taken from a totally ordered set X such that $|X| \geqq N$, then $T \geqq$ $(\lg n/\lg k + O(1))$.*

Proof. The same argument that was twice applied works here as well. Inequality (6.5) of Theorem 6.2 is valid with $K = k$ and $H = 2^{T-1}$. We obtain that $n \leqq \frac{1}{2}H(2K+3)^T \leqq \frac{1}{4}(4k+6)^T$, which yields the result. □

The last result is asymptotically optimal: if processors may store in their local memory k keys, then it is possible to search a table of size n in $O(\lg_{k+1} n + \lg k)$ steps.

8. Conclusion. As mentioned in the introduction, ultracomputers can be seen as a restricted class of EREW paracomputers. Thus, each of the lower bounds is valid for ultracomputers. Consider a network of processors, each directly connected to all the other ones, such that each processor contains one key from the searched table, and one processor contains the searched key. If each processor has a fixed size local memory, then $\Omega(\lg n)$ communication steps are required to perform a search, even if local computations are allowed for free. If local memory is not restricted in size, but only one input value may be transmitted at a time, then the problem can be solved in $O(\lg n/\lg\lg n)$ communication steps. Finally, if there are no restrictions on the type of information that can be transferred in one communication step, then the problem can be solved in $O(\sqrt{\lg n})$ steps.

There are few methods known to prove lower bounds for parallel algorithms, which are not based on fanin arguments. This paper contributes one such new method. It seems to capture two "real-life" problems encountered while writing parallel programs: it is hard to parallelize algorithms with many test and branch operations; and frequent coordination between concurrent processes may offset any gain obtained from concurrency.

This paper also provides a method to generalize lower bounds obtained for comparison based algorithms to less restricted algorithms. In that, we were inspired by the work of Yao [14]. This method can be useful in other settings as well, and in particular can be used to analyse distributed algorithms [8].

A more natural constraint on information transfer would be to restrict the number of bits that can be stored in one memory cell. We believe that our lower bounds are valid in such model too, but the proofs seem much harder to obtain.

The paracomputer models we presented may suffer a few interesting variations. As noted in § 2, the $O(\lg n/\lg p)$ searching algorithm can be implemented on any EREW shared memory parallel machine where one processor has the ability to broadcast messages to all the other processors in constant time (a BEREW machine?). If all the processors share this broadcasting ability (only one broadcast is allowed at a time), then this algorithm can be implemented even in the absence of shared memory. We have here a model of parallelism, corresponding to a bus-oriented architecture. A similar model was studied by Stout [11].

Another natural variation is to assume that conflicting memory accesses do not result in an error, but rather in a busy signal being returned to all but one of the requests; alternatively one may postulate a queuing scheme at the memory.

In a real parallel machine memory is likely to be organized into modules with exclusive access being enforced at the level of the memory module rather than at the level of the memory cell. This suggests that we consider computational models where the number of shared memory cells is restricted, and where the amount of information that can be transferred in one read or write operation is smaller than the content of a memory cell. The work of Baer, Du and Ladner [1], and of Vishkin and Wigderson [13] is a useful start in the investigation of such systems.

9. Acknowledgments. I would like to thank Clyde Kruskal who provided the initial thrust for this work, and Allan Borodin and Chee Yap for their helpful remarks.

REFERENCES

[1] J. L. BAER, H. C. DU AND R. E. LADNER, *Binary search in a multiprocessor environment*, IEEE Trans. Comput., C-32 (1983), pp. 667-676.

[2] A. BORODIN AND J. E. HOPCROFT, *Routing, merging, and sorting on parallel models of computations*, Proc. 14th ACM Symposium on Theory of Computing, 1982, pp. 338-344.

[3] A. BORODIN, *Structured versus general models in computational complexity*, in Logic and Algorithmic, Monographie no. 30 de l'Enseignement Mathématique, Université de Geneve, 1982.

[4] S. COOK AND C. DWORK, *Bounds on the time for parallel RAM's to compute simple functions*, Proc. 14th ACM Symposium on Theory of Computing, 1982, pp. 231-233.

[5] D. M. ECKSTEIN, *Simultaneous memory access*, Res. Rep. TR-79-6, Computer Science Dept., Iowa State Univ., Ames, 1979.

[6] S. GAL AND W. MIRANKER, *Optimal sequential and parallel search for finding a root*, J. Combin. Theory (A), 23 (1977), pp. 1-14.

[7] C. P. KRUSKAL, *Upper and lower bounds on the performance of parallel algorithms*, Ph.D. Thesis, Computer Science Dept., New York Univ., New York, 1981.

[8] NANCY LYNCH, *private communication.*

[9] F. P. RAMSEY, *On a problem of formal logic*, Proc. London Math. Soc., 2nd ser, 30 (1930), pp. 264-286.

[10] J. T. SCHWARTZ, *Ultracomputers*, ACM Trans. Programming Languages and Systems, 2 (1980), pp. 484-521.

[11] Q. F. STOUT, *Mesh-connected computers with broadcasting*, IEEE Trans. Comput., C-32 (1983), pp. 826-830.

[12] M. SNIR, *On parallel searching*, ACM Symposium on Principles of Distributed Computing, 1982, pp. 242-253.

[13] U. VISHKIN AND A. WIGDERSON, *Trade-offs between depth and width in parallel computation*, Proc. 24th IEEE Symposium on Foundations of Computer Science, 1983, pp. 146-153, this Journal, 14 (1985), pp. 303-314.

[14] A. C. YAO, *Should tables be sorted?*, J. Assoc. Comput. Math., 28 (1981), pp. 615-628.