PARALLEL AND DISTRIBUTED ALGORITHMS
M. Cosnard et al. (editors)
© Elsevier Science Publishers B.V. (North Holland), 1989

THE COMPLEXITY OF PARALLEL SEARCH ON COARSE GRAINED NETWORKS[*]

SELIM G. AKL
Department of Computing and Information Science, Queen's University, Kingston, Ontario K7L 3N6, Canada.

FRANK DEHNE
Center for Parallel and Distributed Computing, School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada.

Abstract. We study the time complexity of parallel search in a sorted list of n elements on a coarse grained bounded degree network of $N < n$ processors.

For an infinite sequence of search queries presented to the network and processed in a pipelined fashion, the latency and period of any searching algorithm are $\Omega(D + \log n - \log N)$ and $\Omega(\log n - \log N)$ respectively in the worst case. It is easy to see that these lower bounds are tight. For $n >> N$, the period is $\Omega(\log n)$ which can be achieved sequentially by a single processor. Hence, coarse grained networks are not particularly well suited, *in the worst case*, for the search problem.

On the other hand, we show that a constant period can be achieved on coarse grained networks *probabilistically*. A parallel search algorithm is described which has constant expected period T provided that $n \leq N\,2^{(T/\xi)N}$ (assuming that sequential search in a sorted list of x items can be performed in time $\xi \log x$, for a constant ξ). Therefore, coarse grained networks with $N << n < O(N\,2^N)$ are good, *on the average*, for the search problem.

1 INTRODUCTION

The parallel complexity of searching a sorted sequence $S=(s_1, \dots , s_n)$ of size n for a given element s was studied by Snir [S85] for an N-processor shared memory model of computation. For a variant of the model, where an item of information can be accessed by one processor only at a time, he showed that searching for one element requires at least $\Omega(\log n - \log N)$ steps. (Henceforth, all logarithms are with respect to the base 2.)

In this paper, we study the parallel complexity of searching for the more realistic model of a *bounded degree processor network* of size N. In this model, a set of N processors $P_1, \dots , P_N$ are connected by bidirectional unit-time communication links between pairs of processors such that the graph thus formed is of bounded degree (e.g., the mesh, tree, mesh-of-trees, pyramid, or hypercube architectures [A88]). By contrast with the model studied in [S85], there exists no shared memory in a processor network. Instead, each processor P_i has its own local finite

[*] Research supported by Natural Sciences and Engineering Research Council of Canada under grants A3336 and A9173.

memory M_i (which can solely be accessed by P_i); processors can communicate only by sending messages via the communication links. The distance between two processors P_i and P_j is the minimum number of direct communication links that a message has to traverse in order to travel from P_i to P_j. The diameter *D* of the network is the maximum distance between processors.

The time complexity of a parallel algorithm executed on a bounded degree network consists of the local computation time for the processors and the time for the messages sent via the communication links. As usual we count, for each processor, an arithmetic or comparison operation as well as a transmission of a word of length O(log n) bits to an adjacent processor as an O(1) time operation.

For the sequential model of computation, the worst case complexity of searching a sorted list of size x is ξ log x, for some constant ξ (this complexity is achieved by the binary search algorithm [AHU76]).

When solving the search problem on a bounded degree processor network, the sequence S is distributed among the M_i's such that each receives a sorted subsequence of size n/N. A designated processor receives as input the element to be searched for and, in the worst case, must propagate it through the network to another processor P_i at a distance D away. In addition, P_i has to search its subsequence sequentialy. Consequently, the worst case lower bound on the time required to search is $\Omega(D+\log\frac{n}{N})$, i.e. $\Omega(D+\log n)$ when n >> N. For example, on the hypercube architecture where D equals log N the lower bound is $\Omega(\log n)$, which can be achieved sequentially by a single processor.

One justification for advocating parallel search is to improve throughput in the case of a stream of queries presented to the network in a pipelined fashion. Therefore, we study the searching problem for bounded degree networks within the following more realistic setting which will be referred to as *pipelined search* (see Figure 1):

- An infinite input sequence of search queries is sent, as an *input stream*, to a designated processor of the network, the *input processor.*
- The search queries are processed in a pipelined fashion and the answers are reported back via another processor, the *output processor* (which may coincide with the input processor).

In this setting, the performance of a searching algorithm is measured by

- the *latency*, i.e., the worst case time between an arrival of a query at the input processor and the reporting of the result by the output processor, and
- the *period*, i.e., the average time T between any two consecutive queries. (More precisely, if the time between the arrival of the first and mth queries is $\Delta\tau_m$ then $T = \lim_{m\to\infty} \frac{\Delta\tau_m}{(m-1)}$.)

The obvious lower bounds for period and latency are $\Omega(1)$ and $\Omega(D)$, respectively. Indeed, for N close to n, Dehne and Santoro ([DS87] and [DS88]) have presented algorithms for mesh and hypercube networks, respectively, with O(1) period and O(D) latency. This leads to m queries being processed in time O(m + D). In fact, both of the above algorithms handle the "dynamic" version of the pipelined search problem since they allow not only queries but also "insert" and "delete" commands among the input stream of messages.

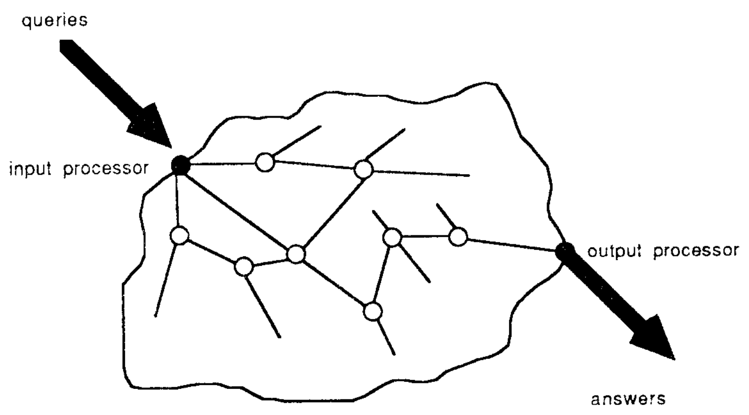

Figure 1: Pipelined Search

However, current attempts at implementing databases on parallel computers, involve *coarse grained* networks. In these networks, each node is a relatively powerful processor with a significant amount of memory. An example of such a system is the Intel iPSC hypercube which consists of a relatively small number (between 8 and 512) of processors with up to 16 MBytes of memory each; for the iPSC/3, currently under development by Intel, a hard disk can even be attached to every sub-hypercube of two or four processors. Under these conditions, the search problem has to be solved for n much larger than N.

For this case we derive, in Section 2, Ω(D + log n - log N) and Ω(log n - log N) lower bounds on the worst case latency and period, respectively. That is, for n >> N the period is Ω(log n) and, hence, m queries are processed in time O(m log n) at best. Since the same performance can be achieved sequentially by a single processor, this result seems to suggest that coarse grained networks are not particularly well suited, *in the worst case*, for the search problem.

On the other hand, we show in Section 3 that a constant period can be achieved on coarse grained networks *probabilistically*. A pipelined search algorithm is described which has a constant expected period T provided that

$$n < N\,2^{\frac{T}{\xi}N} .$$

The algorithm involves a queue (referred to as the *search queue*) to be stored at each processor; this queue has an expected length of at most $L \in O(1)$ provided that

$$n \le N\, 2^{\frac{T\,(L+1-\sqrt{L^2+1})}{\xi} N}.$$

Therefore coarse grained networks with $N << n < O(N\, 2^N)$ are good, *on the average*, for the search problem.

2 The Worst Case Complexity of Pipelined Search on Coarse Grained Networks

In this section we derive worst case lower bounds for pipelined search on a coarse grained bounded degree network with N processors storing $n >> N$ elements. More precisely, we make the following assumptions:

A1) Each processor P_i stores a subsequence S_i of $\frac{n}{N}$ elements.

A2) For each processor P_i, the sequential search time on its subsequence S_i is $\xi \log\frac{n}{N}$.

A3) All elements are stored in sorted order with respect to some total ordering. In particular, each processor P_i has constant time access to the minimum element $\min_i$ and maximum element $\max_i$ of S_i, and for two processors P_i and P_j the two intervals $[\min_i, \max_i]$ and $[\min_j, \max_j]$ are disjoint.

Theorem 1. *With the above assumptions, the pipelined search problem for bounded degree networks has the following worst case lower bounds:*

(a) $\Omega(D + \log n)$ is a worst case lower bound on the ***latency****, and*

(b) $\Omega(\log n)$ is a worst case lower bound on the ***period****.*

Proof. (a) This is identical to the single query case treated in the introduction.

(b) Assume that all queries sent to the network refer to elements contained in the subsequence S_i at one particular processor P_i. In this case, the searching network can also be seen as a single-server queue where requests (i.e., search queries) arrive via the input processor, are queued within the network, and are finally processed by P_i before they leave the system via the output processor (see Figure 2).

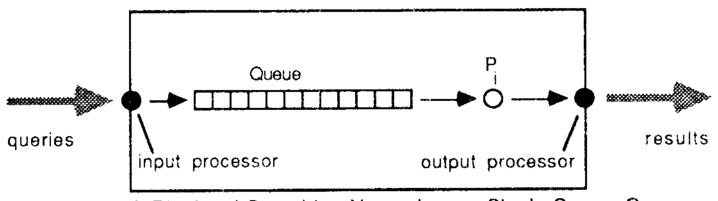

Figure 2: A Pipelined Searching Network as a Single-Server Queue

Let λ and μ denote the average number of arriving queries and departing results per time unit, respectively. That is, $1/\lambda$ is the period and $1/\mu$ is the average time between departing results which is usually referred to as the *service time*. The queueing process is stationary if and only if $1/\mu < 1/\lambda$; i.e., if $1/\mu \geq 1/\lambda$ then for an infinite sequence of queries the length of the queue becomes arbitrarily large ([P84], p. 373). Since the entire searching network has only finite memory available, it follows that if $1/\mu \geq 1/\lambda$ then the network must overflow after a sufficiently long sequence of queries. Hence, the period has to be larger than the service time. Since P_i needs time $\Theta(\log\frac{n}{N})=\Theta(\log n)$, for $n >> N$, to process one query, the service time is at least $\Omega(\log n)$ and, thus, Part b follows. ♦

From Theorem 1 it follows that the total time to perform pipelined search for m queries on a bounded degree network with N processors storing n elements is $\Omega(D + m \log n)$ in the worst case. However, the same problem can be solved on a single processor storing n elements in time $O(m \log n)$ in the worst case; i.e., a single processor is at least as fast or may be even faster than an N-processor network.

3 THE AVERAGE CASE COMPLEXITY OF PIPELINED SEARCH ON COARSE GRAINED NETWORKS

The above worst case lower bounds are based on the fact that all queries may refer to records in the same subsequence S_i stored in processor P_i. Obviously, this is not very likely to happen. Hence, from a practical point of view, it is also important to study the *average case* complexity of pipelined search on coarse grained networks.

In the following, we will present an algorithm to perform pipelined search on a bounded degree network with optimal expected period and latency of $O(1)$ and $O(D_{in} + D_{out}+\log\frac{n}{N})$, respectively; here, D_{in} and D_{out} denote the expected minimum distance from the input and output processors, respectively, to an arbitrary processor P_i. However, this results holds only if n is bounded from above by a function f(N, ...) which grows exponentially with N.

For the given bounded degree processor network, the standard single-source shortest path algorithm ([AHU76], p. 207) determines the shortest path from the input processor to every one of the other processors. These shortest paths define a spanning tree of the network which will be referred to as the *routing tree*. Likewise, the shortest paths from all processors to the output processor define the *report tree*.

The elements are stored such that every processor contains $\frac{n}{N}$ elements in sorted order and the concatenation of these sorted subsequences, defined by the inorder traversal of the processors with respect to the routing tree, is again a sorted sequence. Hence, every query s can be easily routed on its shortest path along the routing tree from the input processor to the processor P_i with $s \in [min_i, max_i]$.

Note that for the above distribution of records, Assumptions A1 to A3 of Section 2 hold.

In addition to the data structure for searching S_i sequentially, every processor P_i maintains two queues for storing pending queries and results which will be referred to as the *search queue* and the *report queue* of P_i, respectively.

The pipelined search algorithm consists of three processes which are executed simultaneously on all processors.

The **routing process.**
Every query s arriving at the input processor is sent via the routing tree to the processor P_i with $s \in [min_i, max_i]$. There, s is inserted at the end of the search queue of P_i.

The **searching process.**
Every processor P_i continuously removes the first query s from its search queue, if one exists, and tests whether $s \in S_i$. The result is inserted at the end of the report queue of P_i.

The **reporting process.**
Every processor P_i continuously removes the first result from its report queue, if one exists, and sends it to its successor on the shortest path from P_i to the output processor (as defined by the report tree).
Every processor receives the results from its predecessors in round robin fashion and inserts them at the end of its report queue.

In the remainder of this section we will study the average-case behavior of the above algorithm under the following assumptions:

A4) For each processor P_i, the time to search its subsequence S_i sequentialy is

$$c = \xi \log\frac{n}{N}. \qquad (1)$$

A5) The arrival of queries at the input port is a Poisson process with period $T \in O(1)$; i.e.,

$$P\{n_t = k\} = \frac{e^{-\alpha t}(\alpha t)^k}{k!} \quad , \alpha = \frac{1}{T} \qquad (2)$$

where n_t is the number of arriving queries within a time period of length t ([P84], p. 210).

A6) The probability is 1/N that the element searched for by a given query resides in a particular processor.

A7) The period T of the query stream is larger than the time to send a query result from one processor to an adjacent one.

Lemma 1.

(a) *The expected length of the search queue of an arbitrary processor P_i is constant if and only if*

$$n < N 2^{\frac{T}{\xi} N} .$$

(b) *The expected length of the search queue of an arbitrary processor P_i is at most a constant L provided that*

$$n \leq N 2^{\frac{T(L+1-\sqrt{L^2+1})}{\xi} N} .$$

(c) *The expected length of the report queue of an arbitrary processor P_i is constant.*

Proof. (a) Consider the search queue at an arbitrary processor P_i (see Figure 3). From Assumptions A5 and A6 it follows that the period of the queries, s, which have to be handled by P_i (i.e., $s \in [min_i, max_i]$) is NT. Hence, the arrivals of these queries at P_i's search queue are a Poisson process with

$$P\{n_t=k\} = \frac{e^{-\lambda t}(\lambda t)^k}{k!} \quad (3)$$

where
$$\lambda = \frac{1}{N\ T} . \quad (4)$$

On the other hand, it follows from Assumption A4 that the time c between two results departing from P_i's search queue is fixed (for given n and N). Hence, the queue at P_i is an M/D/1 queue (see [P84], p. 375). With

$$\rho := \lambda c \quad (5)$$

it follows from (1) and (4) that

$$\rho = \frac{\xi \log n - \xi \log N}{N\ T} . \quad (6)$$

The queue is *stationary* (see [P84], p. 373) if and only if

$$\rho < 1; \quad (7)$$

i.e., if $\rho < 1$ then the expected length of the queue is a constant, otherwise it becomes arbitrarily large. Hence, it follows from (6) and (7) that the expected length of the queue is a constant if and only if

$$n < N 2^{\frac{T}{\xi} N} .$$

(b) In order to prove Part b, we use the fact that the expected length E(q) of an M/D/1 queue is (see [P84], p. 375)

$$E(q) = \frac{\rho(2-\rho)}{2(1-\rho)} . \quad (8)$$

Hence, E(q) is at most a constant L if and only if

$$\rho^2 - 2\rho(L+1)+2L = 0 \quad (9)$$

which is equivalent to

$$\rho = L+1-\sqrt{L^2+1}\text{ , since } \rho<1 \text{ because of (7).} \qquad (10)$$

On the other hand, (6) is equivalent to

$$n = N\,2^{\frac{T\rho}{\xi}N}\text{ ,} \qquad (11)$$

and substituting (10) in (11) we get

$$n = N\,2^{\frac{T(L+1-\sqrt{L^2+1})}{\xi}N}\text{ .} \qquad (12)$$

That is, for this value of n, E(q) is at most a constant L and, hence, Part b follows since for all smaller n, the time c between two outgoing results from P_i decreases (see Equation 1) and therefore also the expected length of the queue.

(c) For every report queue, the period of the input stream is at least T. From Assumption A7 it follows that T is larger than the period of the output stream of the report queue. Hence, the report queue is stationary; i.e., its expected length is a constant (see [P84], p. 372). ♦

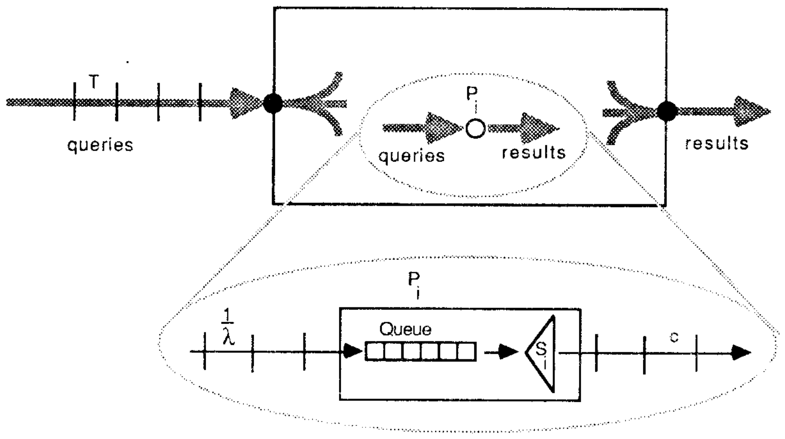

Figure 3: The Searching Process

Theorem 2.

(a) The pipelined search algorithm has constant expected period $T \in O(1)$ which is optimal.

(b) It has optimal expected latency $O(D_{in} + D_{out} + \log\frac{n}{N})$, where D_{in} and D_{out} denote the expected minimum distance from the input and output processors to an arbitrary processor P_i, respectively.

(c) The expected search queue length is is at most a constant L provided that

$$n \leq N\,2^{\frac{T(L+1-\sqrt{L^2+1})}{x}N}$$

Proof. This follows immediately from Lemma 1. ♦

As indicated in Table 1, $L+1-\sqrt{L^2+1}$ is already sufficiently close to 1 for values of L as small as 5.

L	$L+1-\sqrt{L^2+1}$
1	.5858
2	.7639
3	.8377
4	.8769
5	.9010
6	.9172
7	.9290
8	.9377
9	.9466
10	.9501
100	.9950
1000	.9995

Table 1: Some Values of $L+1-\sqrt{L^2+1}$

To illustrate the significance of Theorem 2 in practice, consider for example an Intel iPSC hypercube with N = 512 and L = 5. Table 2 shows the upper bounds on the number of elements per processor, as indicated by Lemma 1, for several values of T/ξ (where T is the desired period and $\xi = \frac{\text{sequential search time in a sorted list of x items}}{\log x}$).

T/ξ	$2^{(.9010 \cdot 512)\ T/\xi}$
1	7.4×10^{138}
.5	2.7×10^{69}
.1	7.7×10^{13}
.05	8.8×10^{6}
.01	24

Table 2: Maximum Number of Elements Per Processor for N=512 and L=5

REFERENCES

[AHU76] A. Aho, J.E. Hopcroft, J.D. Ullman, "The design and analysis of computer algorithms", Addison-Wesley, 1976.

[A88] S.G. Akl, "The design and analysis of parallel algorithms", Prentice-Hall, 1988, in print.

[S85] M. Snir, "On parallel searching", SIAM J. Comput. 14:3, 1985, pp. 688-708.

[DS87] F. Dehne and N. Santoro, "Optimal VLSI dictionary machines on meshes", in Proc. Int. Conference on Parallel Processing, St.Carles, Ill., 1987, pp. 832-840.

[DS88] F. Dehne and N. Santoro, "An optimal VLSI dictionary machine for hypercube architectures", to appear in Proc. Workshop on Parallel and Distributed Computing, Bonas (France), 1988.

[P84] A. Papoulis, "Probability theory, random variables, and stochastic processes", McGraw-Hill, 1984.

[L79] C.E. Leiserson, "Systolic priority queues", Report CMU-CS-79-115, Carnagie-Mellon University, April 1979.

[OB87] A.R. Omondi, J. Dean Brock, "Implementing a dictionary on hypercube machines", *Proc. 1987 Int. Conf. on Parallel Processing*, St. Charles, Ill., 1987, pp.707-709.

[ORS82] T.A. Ottman, A.L. Rosenberg, L.J. Stockmeyer, "A dictionary machine for VLSI", *IEEE Trans. on Computers C-31*, Sept. 1982, pp.892-897.

[SA85] A.K. Somani, V.K. Agarwal, "An efficient unsorted VLSI dictionary machine", *IEEE Trans. on Computers C-34*, Sept. 1985, pp.841-852.

[SL87] A.M. Schwartz, M.C. Loui, "Dictionary machines on cube-class networks", *IEEE Trans. on Computers C-36*, Jan. 1987, pp.100-105.

[SS85] H. Schmeck, H. Schröder, "Dictionary machines for different models of VLSI", *IEEE Trans. on Computers C-34*, 1985, pp.472-475.

[U84] J.D. Ullman, "Computational aspects of VLSI", Computer Science Press, Rockville, MD, 1984.