AN OPTIMAL VLSI DICTIONARY MACHINE FOR HYPERCUBE ARCHITECTURES

FRANK DEHNE and NICOLA SANTORO

SCS-TR-135
April 1988

AN OPTIMAL VLSI DICTIONARY MACHINE FOR HYPERCUBE ARCHITECTURES

FRANK DEHNE [1] and NICOLA SANTORO [2]
Center for Parallel and Distributed Computing
Carleton University
Ottawa, K1S 5B6
Canada

Abstract

In this paper we present a VLSI dictionary machine implemented on a hypercube architecture. The proposed machine consists of two structures, a *snake* and a *broadcast net*, which are both embedded in and operate simultaneously on the same hypercube. All operations (Insert, Delete, Search, Extract Min, and Find Min) can be pipelined with O(1) period, and the response time for Search and Find Min operations is O(log n) and O(1), respectively. Furthermore, the proposed solution is capable of handling duplicate insertions and redundant deletions.

1. INTRODUCTION

A *dictionary* is a device which stores a set of *records* and provides a set of operations on these records, called *dictionary operations*. With each record there is associated a unique *key* k from a totally ordered set K. For simplicity, the record whose associated key is k, will be denoted by *record k* . The standard dictionary operations are

(1) Insert(k) : insert record k in the dictionary

(2) Delete(k) : delete record k from the dictionary

(3) Search(k) : retrieve record k if currently stored; return a negative response otherwise.

In addition to the above operations, the following *priority queue operations* may also be requested:

(4) Find Min: retrieve the current minimum record

(5) Extract Min : delete the current minimum record

Note that only some of these operations require an answer be produced: Search(k) and FindMin; these operations will be called *query operations*.

---

[1]) Research supported by NSERC grant No. A9173

[2]) Research supported by NSERC grant No. A2415

A *VLSI dictionary machine* receives a sequence of instructions (i.e., requests to perform dictionary or priority queue operations), executes the corresponding operations in a *pipelined* fashion, and in case of query operations reports the responses via the I/O port (one designated processor of the hypercube). The *delay* of an instruction is the mimimum number of time units, after the instruction has arrived at the dictionary, necessary before the next instruction can be sent to the machine; the *period* of the machine is defined as the maximum of all instruction delays. For a query operation the *latency* (or response time) is the number of time units elapsed from the time the query arrived to the time a response is produced at the I/O port.

In the literature, many VLSI dictionary machines have been proposed for several architectures: for *trees* [AK85, L79, ORS82, SA85, CCIR86], *cube-connected-cycles* and *shuffle-exchange* networks [SL87], *meshes* [DS87]. As for the *hypercube* architecture, the implementation of a dictionary was discussed in [OB87]; the resulting dictionary is however a concurrent data stucture rather then a VLSI dictionary machine (e.g., no instruction pipelining, every processor is an IO port, etc.). Only some of the existing machines can handle the *redundancy problem*; that is, the problem of ignoring an instruction to insert an element already in or to delete a non-existent element from the dictionary.

In this paper, we propose a VLSI dictionary machine which implements the dictionary and priority queue operations on a hypercube of size $n=2^d$, i.e. a set of 2^d processors P_0, P_1, ..., P_{2^d-1} where processor P_i is connected to processor P_j if the binary representations of i and j differ in exactly one bit position (see Figure 1 for an illustration of a hypercube of size $16=2^4$).

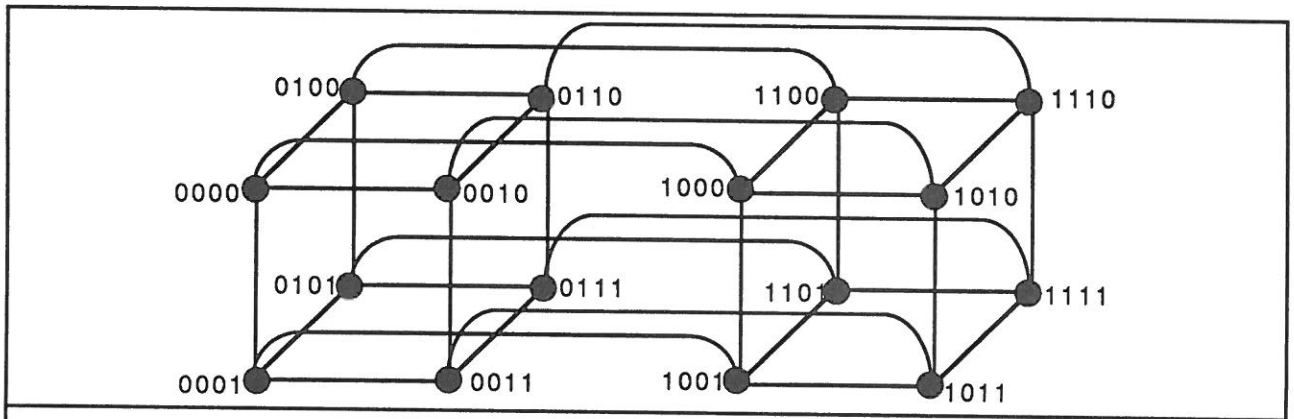

Figure 1: A Hypercube of Size 2^4

The proposed dictionary machine has the asymptotically optimal performance listed in Table 1; it can fully handle the redundancy problem. The basic strategy employed to develop the dictionary is similar to the one used by the authors to obtain optimal dictionary machines for mesh architectures [DS87].

Operation	Performance
Insert	O(1) delay
Delete	O(1) delay
Search	O(1) delay, O(log n) latency
Find Min.	O(1) delay, O(1) latency
Extract Min.	O(1) delay

Table 1. Performance of Proposed Dictionary Machines

2. BASIC STRUCTURE OF THE DICTIONARY MACHINE

The proposed VLSI dictionary machine consists of two known logical structures, a *snake* and a *broadcast net,* which are both embedded in the same physical hypercube and operate on it simultaneously. All incoming search instructions are handed over to the broadcast net, whereas all other instructions are executed by the snake.

The snake is basically the well known systolic priority queue (e.g., [L79], [KL84]). The embedding of the snake in the hypercube is such that every processor is contained in the snake exactly once. All Insert, Delete, FindMin , and ExtractMin instructions are sent to the processor which contains the minimum element; this processor is called the I/O processor of the snake. Note, that if only the snake is operated on the hypercube, Insert, Delete, Extract Min, and Find Min can be performed with O(1) delay and latency, respectively.

The broadcast net is a systolic structure whose function is the handling of Search instructions. Its topology consists of two acyclic directed graphs, G_1 and G_2, where

(a) G_1 is a spanning graph of the mesh with only one *source* ;

(b) G_2 is a subgraph of the mesh with only one *sink*, and whose *sources* coincide with the *sinks* of G_1.

(c) The source of G_1 and the sink of G_2 coincide; this processor is called the I/O processor of the broadcast net.

Each Search(k) instruction is sent to the I/O processor of the broadcast net and processed as follows. The I/O processor (the sole source of G_1) will "broadcast" a message <"broadcast",k,v,r> through G_1, where v is a Boolean value denoting whether record k has been found, and r is a field containing record k if found. The sinks of G_1 will then start a "reverse broadcast" process by sending <"reverse broadcast",k,v,r> messages through G_2; this process has the final effect of collecting at the I/O port (the sole sink of G_2) either record k (if it is in the dictionary) or a negative acknowledgment (v=0).

The only additional constraint on the structure of the broadcast net is that, if more than one message is received on the same graph by the same processor at the same time, they must all contain the same search key k.

Note, that if only the broadcast net is operated on the hypercube, then (i) the latency for Search instructions is d_1+d_2 time units where d_1 [d_2] denotes the distance between the source of G_1 [a source of G_2] and a sink of G_1 [the sink of G_2] and (ii) the delay necessary after each Search instruction is one time unit and; hence, the period of the broadcast net is one time unit.

When simultaneously embedding and operating the snake and broadcast net on the same hypercube, several factors must be taken into consideration; in fact, most embeddings would not archieve the desired performance and, even worse, would not correctly perform the desired operations. The following factors have to be taken into consideration:

- Processor and line contention may occur if both processes try to use the same resource (a processor or communication line of the hypercube) at the same time.
- The simultaneous execution of Insert and Delete instructions by the snake process may cause problems for the correct execution of Search instructions handled by the broadcast net. In particular, the search process must (1) take into account any update instruction started before it, even if the snake has not yet comleted the update instruction, (2) consider update instructions which are currently executed by the snake and may cause other data (maybe the data which are currently searched for) to be shifted within the snake, and (3) disregard all update instructions started after it.

In [DS87] a solution to these problems for the concurrent execution of a snake and broadcast net on a mesh architecture has been introduced. It is easy to see that this solution can also be applied to the hypercube; it can be summarised as follows:

- If the embedding of the broadcast net and snake are edge disjoint then processor and line contention does not occur; otherwise, the broadcast net and snake must be operated in alternating phases which, however, slows down the performance of the dictionary machine by a factor of two.
- To ensure the correct execution of Search instructions some additional information must be kept at each PE. The amount of information is proportional do Δ which is defined as follows:
  For each processor P let *DIST(P)* denote the number time steps necessary for a search message to travel from the I/O processor to P (within the graph G_1 of the broadcast net), and
  $\Delta(P) := \max\{ |t(P)-t(P')| : P \text{ and } P' \text{ are directly connected by an edge in the snake}\}$,
  then Δ is the maximum $\Delta(P)$ of all processors P.

Δ is strictly dependent on the interconnection between snake and broadcast net when embedded in the hypercube. In fact, given an embedding, the value Δ and whether the embedding is edge disjoint or shared completely characterize the interference between snake and broadcast net.

3. AN EDGE DISJOINT EMBEDDING OF BROADCAST NET AND SNAKE IN THE HYPERCUBE

In this section we will describe an edge disjoint embedding of the broadcast net and snake on a hypercube which also guarantees Δ=3. This embedding induces a VLSI dictionary machine on a hypercube with the asymptotically optimal performance listed in Table 1.
We will first describe the embedding of the broadcast net and then the embeding of the snake.

The global structure of the broadcast net embedding is shown in Figure 2a. The hypercube is split into 2^{d-4} sub-hypercubes of size 2^4. The processors in each sub-hypercube are numbered from x..x0000 to x..x1111.
For the remainder, two processors Pi and Pj will be called *k-neighbors*, $1 \le k \le d$, if the binary representations $i_1 i_2 .. i_d$ and $j_1 j_2 .. j_d$ of i and j, respectively differ exactly in the k^{th} bit.
The processor $P_{1..10100}$ is considered the I/O processor of the broadcast net. After receiving a Search(k) instruction, a copy of this instruction is first sent to the d-neighbor of the I/O processor. Then, both processors send a copy to their (d-1)-neighbora. This process is iterated a total number of d-4 times; i.e. for k=d,..,5 all processors which have so far received the message send a copy to their k-neighbor. The result of this process is that after d-4 steps everyy processor $P_{x..x0100}$ has received a Search(k) message. Now, all processors $P_{x..x0100}$ in parallel send the Search(k) message through the broadcast net of their sub-hypercube of size 2^4 as shown in Figure 2b. The messages arrive at the 2^{d-3} sinks, two for each sub-hypercube, at the same time. The sinks then initiate the reverse broadcast process which is exactly the same, but reverse.
In order to implement the pipelining of broadcast processes for Search instructions which subsequently arrive at the I/O processor, every processor $P_{x..x0100}$ stores incoming search messages in a queue of length d-4 (with positions numbered from 5 to d) as shown in Figure 2c. An incoming message from an (i+1) neighbor is stored at position i; for the I/O processor i=d. At every step, each message stored at position k, $5 \le k \le d$, is sent to the k-neighbor and then all messages are shifted one position towards the lower numbered end of the queue. A message which was stored at position 5 is broadcasted through the sub-hypercube of size 2^4.
The mechanism ensures that (1) search processes are correctly pipelined and (2) the broadcast of a Search(k) message through the sub-hypercubes of size 2^4 starts at all sub-hypercubes at the same time. (2) and the timing in subhypercubes indicated in Figure 2b provide that the additional constraint on the structure of the broadcast net is not violated, i.e. if more than one

message is received on the same graph by the same processor at the same time then they contain the same search key.

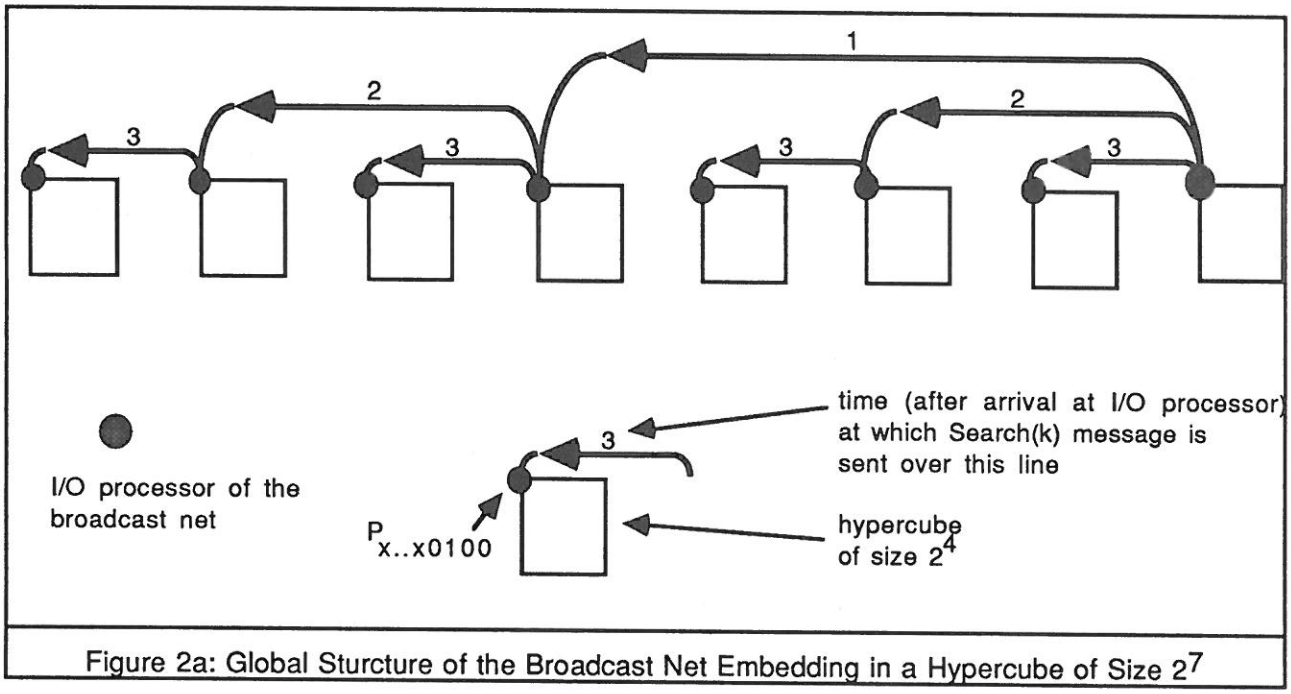

Figure 2a: Global Sturcture of the Broadcast Net Embedding in a Hypercube of Size 2^7

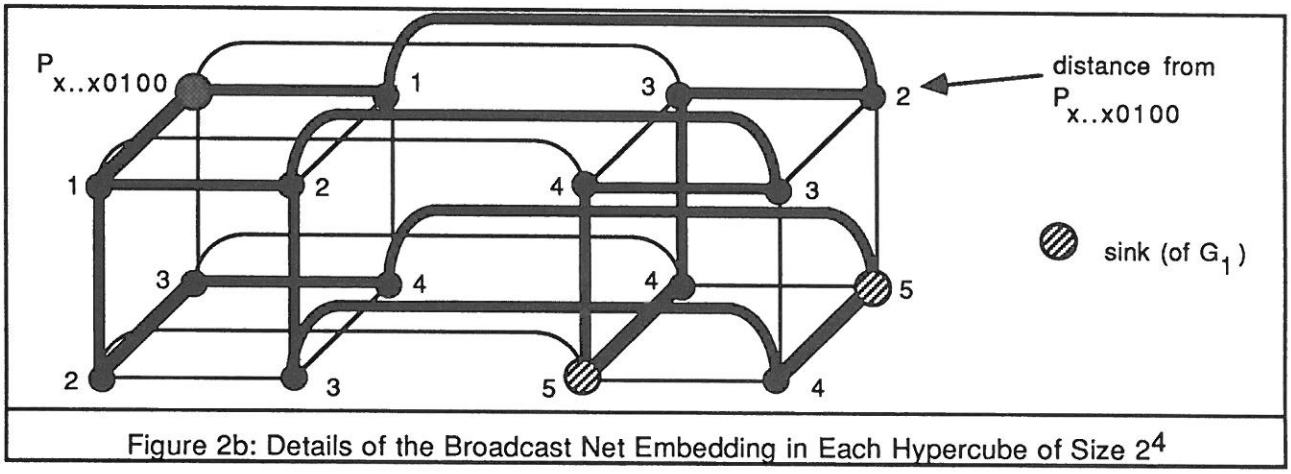

Figure 2b: Details of the Broadcast Net Embedding in Each Hypercube of Size 2^4

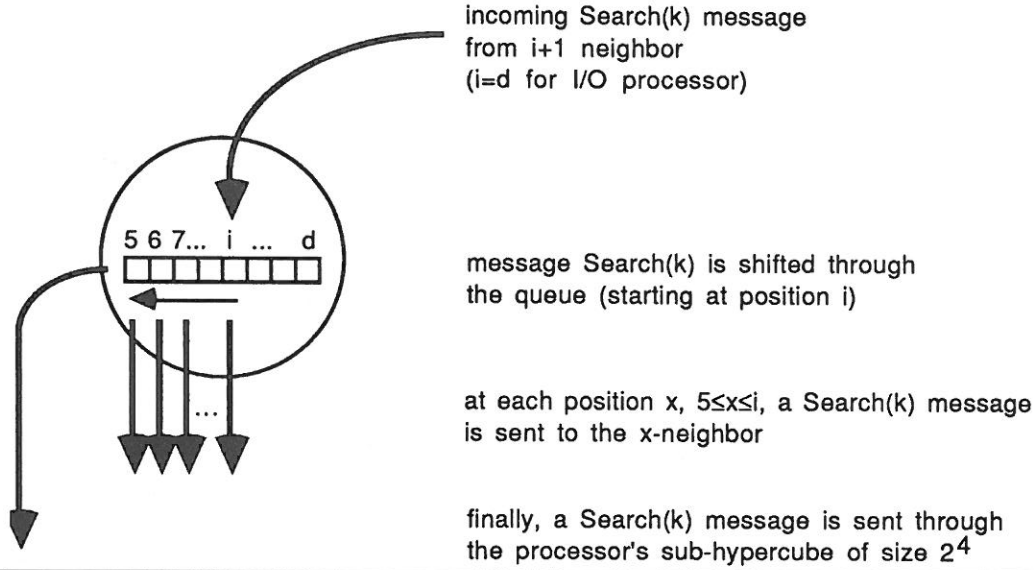

Figure 2c: Handling of Search(k) Messages by Each $P_{x..x0100}$

The structure of the snake is shown in Figures 3a and 3b. Figure 3b shows the snake in each sub-hypercube of size 2^4. The snake has two endpoints $P_{x..x0000}$ and $P_{x..x0010}$ which are labeled A and B, respectively. The snakes in the sub-hypercubes are then connected to form one snake that contains all processors as shown in Figure 3a. The processor $P_{1..10000}$ is the I/O processor of the snake. Note, that the direction in which the snakes in the sub-hypercubes are operated alternates between the sub-hypercubes.

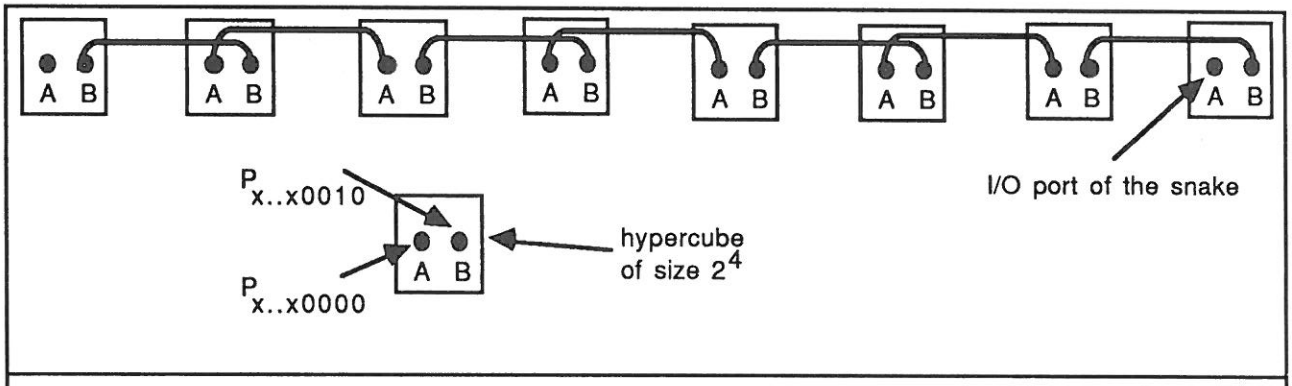

Figure 3a: Global Structure of the Snake Embedding in a Hypercube of Size 2^7

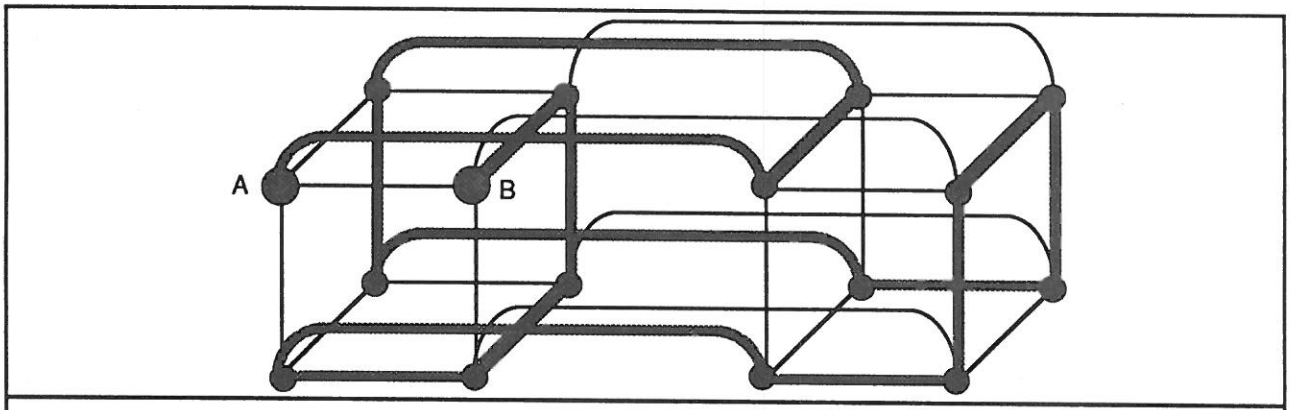

Figure 3b: Details of the Snake Embedding in Each Hypercube of Size 2^4

In order to prove the correct concurrent execution of the snake and broadcast net, the following two properties are easily established:

- Within each sub-hypercube of size 2^4, the embedding of the snake and broadcast net are edge disjoint (compare Figures 2b and 3b) and the "blobal wireing", i.e. the edges between sub-hypercubes, of the snake and broadcast net are also edge disjoint.
  Therefore, the embeddings of snake and broadcast net are edge disjoint.
- The broadcast of a Search(k) message starts at processor $P_{x..x0100}$ of each sub-hypercube at the same time. Therefore, all edges of the global wireing of the snake only connect processors with the same distance from the I/O port with respect to the broadcast net. From Figures 2b and 3b it is also easy to observe that within each sub-hypercube the distance (in the broadcast net) of two processors from $P_{x..x0100}$ which are connected by a snake edge is at most 3.
  Hence, for the above embedding of the snake and broadcast net, $\Delta=3$.

Finally, since in the broadcast net the distance of the I/O processor from the sinks (of G1) is $d+1 = \log(n)+1$, the asymptotically optimal performance results liste in Table 1 follow immediately.

REFERENCES

[AK85] M.J. Atallah, S.R. Kosaraju, "A generalized dictionary machine for VLSI", *IEEE Trans. on Computers C-34*, 2 (Feb. 1985), pp.151-155.

[CCIR86] J.H. Chang, M.J. Chung, O.H. Ibarra, K.K. Rao, "Systolic tree implementation of data structures", *Proc. 1986 Int. Conf. on Parallel Processing*, St. Charles, Ill., 1986, pp.669-671.

[DS87] F. Dehne, N. Santoro, "Optimal VLSI dictionary machines on meshes", *Proc. 1987 Int. Conf. on Parallel Processing*, 1987, pp.83-840.

[KL84] M.R.Kramer, J. v.Leeuwen, "Systolische Berechnungen und VLSI", *Informatik Spektrum* 7, 1984, pp.154-165

[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