Self-Organizing Lists on the Xnet

Lynn M. Stauffer
University of California, Irvine
Irvine, CA 92717
stauffer@ics.uci.edu

Daniel S. Hirschberg
University of California, Irvine
Irvine, CA 92717
dan@ics.uci.edu

Technical Report 92-81

September 28, 1992

ABSTRACT

The first parallel designs for implementing self-organizing lists on the Xnet interconnection network are presented. Self-organizing lists permute the order of list entries after an entry is accessed according to some update heuristic. The heuristic attempts to place frequently requested entries closer to the front of the list. This paper outlines Xnet systems for self-organizing lists under the move-to-front and transpose update heuristics. Our novel designs can be used to achieve high-speed lossless text compression.

1. Introduction

We describe the first parallel designs for self-organizing lists on the Xnet interconnection network. Self-organizing lists permute the order of list entries after an entry is accessed, attempting to place more frequently requested entries closer to the front of the list. Previous work in the parallel domain is limited to the implementation of self-organizing lists on the systolic array. This paper provides Xnet designs for self-organizing lists under the move-to-front and transpose permutation heuristics. These high-speed architectures can be used to perform lossless text compression.

The list organizing strategies of interest in this paper are move-to-front and transpose (see [HH85] for a survey of self-organizing linear search). The move-to-front strategy moves the accessed record to the front of the list, shifting all records previously ahead of it back one position. The transpose heuristic permutes the list by exchanging the accessed entry with the one immediately before it in the list. After reaching a steady state, where many further search requests are not expected to significantly impact the expected search time, the expected access cost is less for transpose than for move-to-front, but the convergence time or number of accesses required to reach a steady state is greater for transpose than for move-to-front [HH85]. There are applications for which move-to-front and transpose outperform each other. For any particular application, simulations are necessary to determine the superior heuristic.

The model of parallel computation used in this work is the data parallel Xnet. The Xnet interconnection network provides a modified mesh-connected structure suitable for rapid communication among neighboring processing elements (PEs). Switches, located between each pair of PEs, permit vertical, horizontal, or diagonal connections. The connections wrap around meaning that switches are located between the top and bottom row of the mesh and between the extreme left and right

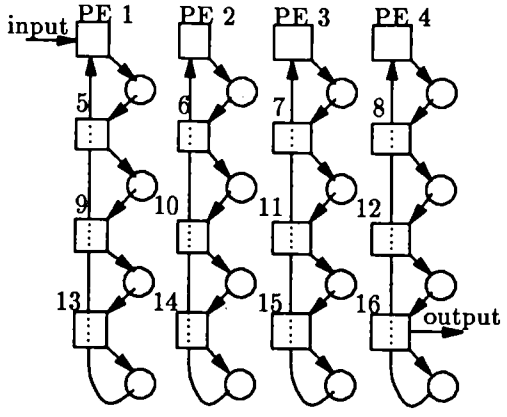

Figure 1

Xnet interconnection network with switches set North-to-South

columns. During each system cycle, the switches are identically set throughout the system and computation proceeds synchronously. Each PE is connected to two of its nearest neighbors and communication proceeds in a single direction. For instance, if the switches are set to provide North-to-South connections, a single PE receives input from the processor above it in the mesh and transmits data to the processor below it. Each PE has its own local memory and is assigned a unique identification number. An Xnet on 16 processors (4 by 4 mesh) with the switches set North-to-South is pictured in Figure 1. The wrap around connections in each column link the bottom PE to the top PE. For the systems described in this paper, input enters the Xnet at PE 1 and output exits at PE N^2 (assuming an Xnet of dimensions N by N).

Our Xnet architectures for self-organizing lists can be used to obtain high-speed text compression systems. Text compression attempts to remove redundancy from data and thereby increases the density of transmitted or stored data. Traditionally, there has been a tradeoff between the benefits of employing compression versus the computational costs related to encoding and decoding. Parallelism

represents a means for increasing the speed of performing compression. Move-to-front and transpose compression maintain a dynamic list of words (to be encoded by their list position) using the move-to-front and transpose self-organizing list strategies, respectively [BSTW86, E87, R87, HC87]. Systolic array implementations for transpose and move-to-front compression have been described [SH92, TW89].

Related work on systolic array architectures for self-organizing lists is presented in the next section. Our Xnet systems for move-to-front and transpose are given in Sections 3 and 4. Text compression and the application of our Xnet systems is discussed in Section 5. Section 6 proposes areas for further investigation.

2. Related Work on the Systolic Array

Previous work on parallel implementations of self-organizing lists is limited to the systolic array model [SH92, TW89]. A systolic array consists of a linearly-connected collection of synchronous rudimentary processing elements (PEs). VLSI implementation issues regarding input/output pin requirements have forced the investigation of two major algorithmic variants. The simpler procedure permutes a byte-level fixed-length list of symbols and the other approach permits arbitrary-length list entries. Arbitrary-length list entries that must be communicated among PEs force an arbitrary number of input/output pins on each PE.

Thomborson and Wei investigate systolic array implementations of move-to-front [TW89]. For the simpler byte-level fixed-length scheme, each PE stores a single character. The input stream enters the pipe and matches are detected between the input characters and the bytes stored in the PEs. For the arbitrary-length scheme, Thomborson and Wei examine various alternatives, such as placing a limit on the length of words and hashing.

Systolic array architectures for transpose have also been described for both the byte-level and arbitrary-length variants which operate at speeds commensurate

with the move-to-front systems [SH92]. An design combining systolic arrays with trees improves the through delay from linear to logarithmic.

These systolic systems for move-to-front and transpose can be used to perform high-speed lossless text compression. Sequential compression systems operate at data rates ranging from 10 to 30 Kbytes/second. Systolic schemes operate at much higher data rates of roughly 40 Mbytes/second.

3. Self-Organizing Lists of Fixed-Length Entries on the Xnet

As pointed out in the previous section, a parallel implementation may have difficulty allowing non-fixed length words to travel between processing elements because of the potentially unreasonable pin requirements. Xnet systems for self-organizing lists which permute a list of fixed-length list entries are described in this section. Approximations for the more general list of arbitrary-length records are addressed in Section 4.

3.1. Move-to-Front

To transmit word w on the Xnet using the move-to-front heuristic, w is compared to the list entries of successive processors. At PE 1 (the front of the list) w overwrites the current list entry which is transmitted to PE 2. Next, the list entry of PE 2 is overwritten by the previous entry of PE 1. List entries continue to cascade down the list until w matches the list entry and the final entry is updated. By depositing the input character in the first processor as it enters the pipe and then cascading previous processing element contents down the array, the move-to-front behavior is realized. For convenience, the move-to-front system on the Xnet is referred to as XMTF.

For a list of length N^2, XMTF consists of N^2 processing elements (PEs). PE i stores the list entry which is currently in position i in the list (referred to as $entry_i$). The input enters the Xnet at PE 1 and exits at PE N^2.

Each step of XMTF consists of updating the list and checking for matches between the input and the stored entries. Each step is carried out in 2 phases. During Phase 1 the Xnet switches are set to connect NorthWest-to-SouthEast (NW-SE) PEs. PE kN, where $k \geq 1$, sends 4-tuple (w, e, p, f) to PE $kN+1$. w is the input being accessed in the list, e is the current contents of $entry_i$, p is the list position of w (p is used to perform compression) and f is a bit flag which is set if $entry_i$ is to be cascaded to PE $i+1$. In Phase 2, the Xnet connections are changed to link processors East-to-West (E-W). PE i, where i is not a multiple of N, transmits 4-tuple (w, e, p, f) to PE $i+1$. If $f = 1$, PE i exchanges $entry_i$ and e. If w matches the new entry stored in e (equivalently the previous contents of $entry_i$) then PE i sets $f = 0$.

3.2. Transpose

To transmit word w on the Xnet using the transpose heuristic for self-organizing lists, w is compared to list entries of successive processors. If w matches the list entry in PE i then the list is updated by transposing the list entry (w) of PE i with the list entry in PE $i-1$. Several matches can be detected simultaneously on the Xnet. To avoid the contention associated with updating the list among neighboring processors which have concurrently detected matches, our Xnet architecture for transpose (referred to as XTR) allows input packets to enter the mesh only on every other system cycle. First, the word list is updated and, later in the cycle, word matches are detected.

For XTR on a list of length N^2, the Xnet consists of N^2 PEs. PE i stores the i^{th} list record and a copy of the input word PE i considered on the previous system cycle. The list entry is referred to as $entry_i$ and the prior input word as $oldw_i$.

In Phase 1, the Xnet switches are set NW-SE. PE kN, where $k \geq 1$, sends 3-tuple (w, e, p) to PE $kN+1$. As in XMTF, w is the input being accessed in

the list, e is $entry_i$ that may be needed for transpose update, and p is the list index of w used for performing compression. Switches are changed to reverse the diagonal connections of Phase 1 to SE-NW links in Phase 2. PE $kN+1$, $k \geq 1$ sends bit m_{kN+1} to PE kN. m_{kN+1} is a bit flag which is set if PE $kN+1$ detected a match in the previous system cycle. At the start of Phase 3, the Xnet alters the connections to W-E links. PE i, where i is not a multiple of N, sends 3-tuple (w, e, p) to PE $i+1$. In Phase 4, the switches link E-W and PE $i+1$, where i is not a multiple of N, transmits m_{i+1} back to PE i. If m_{i+1} is set then PE i (for all i) overwrites $entry_i$ with $oldw_i$. If w matches $entry_i$ then PE i sets $m_i = 1$ and (if $i > 1$) overwrites $entry_i$ with input e. Finally, PE i overwrites $oldw_i$ with w. Contention is avoided as a result of restricting input to every other cycle.

4. Self-Organizing Lists of Arbitrary Words on the Xnet

The designs of Section 3 included only fixed-length words in the list. The more general word list scheme permits words of arbitrary lengths to be present. A straightforward extension of our previous designs allows arbitrary words. However, the packets that travel through the mesh may lead to unreasonable VLSI implementation requirements. One simple solution to the problem of unbounded pin requirements is to place a bound on the maximum allowable word length. This bound enforces a limit on the pin requirements. The appropriate maximum word length is dependent on the application.

In order to avoid the potential VLSI issues, an approximation using a hard-wired hash table to map arbitrary words onto an 8-bit byte has been considered [SH92, TW89]. This single byte is input into the system and handled as a fixed-length word. Implementing this hashing approach on our Xnet architectures is straightforward.

5. List Compression

The Xnet architectures of the previous sections can be used to perform high-speed text compression. Text compression systems remove data redundancy in communicated and stored data and increase the effective capacities of communication and storage devices. Dictionary compression algorithms function by replacing blocks of input with references to earlier occurrences of identical data [BCW90]. Systolic implementations have been developed for several variants of the general dictionary scheme. Parallel dictionary methods are surveyed in [SH91].

This paper considers a class of compression algorithms which maintains a sequential list of words using a self-organizing heuristic so that frequently accessed words appear near the front of the list. We refer to this collection as *list compression* methods under a particular update heuristic.

A list compression method uses a self-organizing data structure to maintain a list of source messages and an encoding of the integers to compress list indices. A variable-length encoding of the integers, such as Elias codes, Fibonacci codes, and start-step-stop codes, or a non-codeword based coding such as arithmetic coding can be used to compress list positions [BCW90]. To compress a word, it is located in the dynamic word list and encoded by its list position. After a word has been referenced, the list is reorganized appropriately. In move-to-front compression, the encoded word is removed from its current position and placed in the first list position. In transpose compression, the encoded word is exchanged with the contents of the preceding list entry. By directing the encoded list position to a fixed-to-variable coder (located in a special-purpose processor at the tail of the array), the output is further compressed by assigning short codewords to positions near the front of the list. Arithmetic coding assigns shorter encodings to more frequently occurring list indices.

Parallel list compression is described by the following general paradigm. Encoder and decoder maintain identical word lists using the same list-organizing heuristic. After a word is used or matched it is encoded by its list position and the list is permuted according to the update strategy. The Xnet systems for self-organizing lists can be used to perform parallel list compression. During encoding, the match detection step is augmented so that when a match is recognized, the input character is replaced by the identification number of the matching processor. In the notation of Section 3, if PE i detects a match between the input w and $entry_i$ then w is encoded by list index i (p is set to i).

6. Future Work

Although we have outlined the design of several Xnet implementations for high-speed applications, extensive empirical evaluation of our methods are needed to fully describe their behavior. For text compression applications, comparative studies of speed and compression effectiveness are required to determine the advantages of these systems.

The approximations for self-organizing lists of arbitrary-length records are limited. Alternative schemes are necessary to better emulate general dynamic lists.

This work represents the extension of parallel self-organizing lists to a novel parallel model. Further investigation of self-organizing lists and other dynamic data structures on other models of massively parallel systems are warranted. Move-ahead-k, for example, is a self-organizing heuristic which compromises between the extremes of move-to-front and transpose. Move-ahead-k transfers the accessed entry k positions forward. Parallel versions of this and other heuristics require further study.

References

[BCW90] Bell, T. C., Cleary, J. G., and Witten, I. H. *Text Compression*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.

[BSTW86] Bentley, J. L., Sleator, D. D., Tarjan, R. E., and Wei, V. K. A locally adaptive data compression scheme. *Commun. ACM 29*, 4 (April, 1986), 320–330.

[E87] Elias, P. Interval and recency rank source coding: two on-line adaptive variable-length schemes. *IEEE Trans. Inform. Theory IT-33*, 1 (Jan., 1987), 3–10.

[HC87] Horspool, R. N., and Cormack, G. V. A locally adaptive data compression scheme. Technical Correspondence. *Commun. ACM 30*, 9 (Sept., 1987), 792–794.

[HH85] Hester, J. H. and Hirschberg, D. S. Self-organizing linear search. *ACM Comp. Sur. 17*, 3 (Sep., 1985), 295–311.

[R87] Ryabko, B. Y. A locally adaptive data compression scheme. Technical Correspondence. *Commun. ACM 30*, 9 (Sept., 1987), p. 792.

[SH91] Stauffer, L. M. and Hirschberg, D. S. Parallel data compression. Technical Report 91-44. Info. and Comp. Sci. Department, University of California, Irvine.

[SH92] Stauffer, L. M. and Hirschberg, D. S. Transpose coding on the systolic array. In *Proceedings IEEE Data Compression Conference*, Snowbird, Utah, IEEE Computer Society Press, 1992, pp. 162–171.

[TW89] Thomborson, C. D. and Wei, Belle W.-Y. Systolic implementations of a move-to-front text compressor. In *Proceedings 1989 ACM Symposium on Parallel Algorithms and Architectures*, Sante Fe, New Mex., ACM, New York, 1989, pp. 283–290.