Table 1. Performance comparison of systolic arrays for LU-decomposition

Fig.	# *PE*	δ	Diagonal connection	Preloading	Execution	Draining
1	n^2	1	No	n	$3n-2$	$n-1$
2	$n(n+1)/2$	1	No	0	$3n-2$	$n-1$
3	$n^2+(n-1)^2$	1	Yes	0	$3n-2$	1
4	n^2	3	Yes	$n-1$	$3n-2$	1
5	$n(n+1)/2$	2	No	0	$3n-2$	1

Hence it needs more hardware cost. But there is no need for preloading time of A_{ij}, hence this design has the minimum computation time.

The array size of Fig. 4 is equal to n^2, but there is no need for processor reprogramming. The function of *PE*s are fixed all the time, i.e. the first column *PE*s perform division, the first two *PE*s only pass data, and the inner *PE*s perform the multiply-and-add operation. Its disadvantage is that the pipelining period is equal to 3.

The function of each *PE* in Fig. 5 may change at one given moment and its pipelining period is equal to 2. But it needs only $n*(n+1)/2$ *PE*s and there is no need for preloading time; it also has the advantage of diagonal connection.

6. Conclusion

Conventional design of systolic arrays is based on the mapping of an algorithm onto an interconnection of processing elements. This mapping is done in an *ad hoc* manner. In this paper we present a notation of the generating function which is a mathematical formal approach to represent systolic arrays. With the properties of linear algebra, it supports the transformation between various equivalent systolic designs.

Y-C. HOU* and J-C. TSAY
Department of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, 30050, R.O.C.

* To whom correspondence should be addressed.

References

1. M. C. Chen, Synthesizing systolic designs, *International Symposium on VLSI Technology, System and Applications*, pp. 209–215 (1985).
2. Y. C. Hou and J. C. Ysay, An algebraic model for representing equivalent designs of systolic arrays. *Proc ISMM International Symposium*, Florida, pp. 163–166 (1988).
3. K. Hwang and Y. H. Cheng, VLSI computing structures for solving large scale linear system of equations, *Proc. Parallel Processing Conference*, pp. 217–227 (1980).
4. H. T. Kung, Highly concurrent systems, *Introduction to VLSI System*, edited C. A. Mead and L. A. Conway, Reading, MA; Addison-Wesley (1980).
5. S. Y. Kung, On supercomputing with systolic wavefront array processors, *Proc. IEEE*, **72** (7), 867–884 (1984).
6. S. Y. Kung, *VLSI Array Processors*. Prentice-Hall (1988).
7. G. J. Li and B. W. Wah, The design of optimal systolic arrays. *IEEE Trans. on Computers*, **C-34**, 66–77 (1985).
8. D. I. Moldovan, On the design of algorithms for VLSI systems. *Proc. IEEE*, **71** (1),m 113–120 (1983).
9. J. C. Tsay and Y. C. Hou, Generating function and equivalent transformation for systolic arrays, to appear in *Parallel computing*, North-Holland.
10. S. Yuan, *An Algebraic Notation for the Design and Verification of Systolic Arrays*, M.S. Thesis, National Chiao Tung University, Hsinchu, R.O.C. (1987).

Reverse Chaining for Answering Temporal Logical Queries

A possible structure for the past data of a partitioned temporal database is reverse field chaining. Under this technique field versions are chained by descending time. Exact analysis derives the expected number of block accesses when logical queries against a partitioned temporal database with reverse chaining are satisfied. Numerical results are given.

Received November 1989, revised November 1990

1. Reverse chaining

A very efficient structure for answering queries based on secondary key values against a temporal database is *reverse chaining*, according to which field versions are chained by decreasing time. The motivation of the present work is a previous effort on the performance analysis of such systems,[1] where a formula was given deriving the cost of answering logical queries by using reverse chaining. However, the analysis was approximate and resulted in a pessimistic evaluation. Here, we give an improved mechanism for traversing many reverse chains by using some sort of parallelism instead of a one-a-time chain processing.

Suppose that the original layout of a record from a personel file of a conventional database is:

EMP(empno,name,position,salary).

This file in database applications with time support should have a different layout. If field version chaining is applied and the user queries concern the fields 'position' and 'salary' then the previous layout should be:

EMP(empno,name,position,*p_chain*,
salary,*s_*chain,time)

Figure 1 is a graphical representation of this layout.

Suppose that the following logical query is posed in this file form: 'Retrieve the salaries and the positions of empno = x for the past three years'. This logical query is decomposed in two simple queries as: 'Retrieve the salaries of empno = x for the past three years' and 'Retrieve the positions of empno = x for the past three years'.

In the absence of any indexes for accessing history data the relevant records should be retrieved by following the two chains. In Ref. 1 it was assumed that the two chains are followed independently from each other. Therefore the total cost was considered to be the sum of the costs for answering each single query, where the cost of each single query is given by the ubiquitous formula by Yao[3] for the expected number of block accesses when a number of fixed length file records is randomly selected.

The two chains of records may pass through the same blocks. If great buffer space is available then block accesses may be saved by navigating simultaneously through the chains instead of searching the chains one after the other. These savings are possible because the past data file, in essence, may be considered as a sequential heap with the attribute 'time', standing for 'valid time from' or 'transaction time', as the primary key record.

The term simultaneous navigation needs some explanation. Every block appended to the heap file has greater address than the predecessor blocks ones'. This block, also, contains records with 'time' field values greater than the corresponding values of the previous blocks. The search in one chain is performed by following the pointer to the next version in the chain (previous in time). The chain to be processed next will be chosen by following the rule: pick the chain pointing to a block with address greater than the address pointed by the other chain. Evidently, it is not certain that any two consecutive searches do correspond to the same chain. If the block to be fetched resides already in main memory, because it belongs to the other chain too, then no I/O cost is paid. Instead, negligible CPU cost is paid to examine the records of the specific block, read the two chain pointers and decide which one to follow. This method is similar to the traversing of multilist files in parallel as reported by Claybrook[2]. In addition if the cost has to be computed in advance for taking decisions on query optimization, then the cost of independent chain search should be replaced by the cost of simultaneous chain search. This is the contribution of our work.

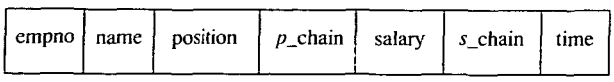

Figure 1. Temporal database record layout.

In Figure 2 the first (second) chain connects versions of the field 'position' ('salary'). The first pointer of the first (second) chain points to block A (B). Block A instead of block B is retrieved since its address is greater. The second pointer of the first chain points to block C. Therefore, block B is retrieved since its address is greater than the address of block C. The second pointer of the second chain points to block C; block C is accessed. Searching will be continued by following again the first chain because the appropriate pointer points to block D with address greater than the address of block E pointed by the pointer of the second chain. Thus block D is retrieved. Then block E is retrieved as pointed by the pointer of the second chain and the method continues in this way.

Table 1. Cost examples for answering two logical queries

r_1, r_2	5, 3	5, 4	5, 5	10, 3	10, 4	10, 5
Cost A	7.44	8.36	9.22	11.20	12.08	12.93
Cost B	6.82	7.50	8.15	10.01	10.52	11.02

Table 2. Cost example for answering three logical queries

r_is	8	9	10	8, 9	8, 9, 10
Cost A	6.94	7.65	8.32	14.58	22.91
Cost B	—	—	—	11.93	15.29

2. Analysis for reverse chaining

Suppose that the past data consist of R fixed length records which occupy B blocks. The block capacity is $bc = R/B$. First, consider that the logical query may be decomposed in two simple ones. Suppose, also, that each simple query is satisfied by r_i records ($1 \leqslant i \leqslant 2$). These r_i records reside in b_i blocks in optical storage. The expected number of block accesses is equal to Ref. 3:

$$b_i = B * \left[1 - \frac{\text{comb}(R - bc, r_i)}{\text{comb}(R, r_i)}\right],$$

where $\text{comb}(a, b)$ is the a-choose-b combination.

However, to the author's knowledge, derivation of the probability distribution of the number of block accesses for retrieving a specific number of records lacks in literature. This distribution is:

$$\text{P}(b_i) = \frac{\text{comb}(B, b_i)}{\text{comb}(R, r_i)} \sum_{\substack{\sum n_j = r_i \\ n_j \neq 0}} \prod_{j=1}^{b_i} \text{comb}(bc, n_j).$$

This formula is explained as follows. The r_i records are distributed in b_i blocks, where b_i ranges from 1 to r_i. The product Π consists of b_i combinations. Each combination represents the number of ways that n_j records may be selected out of the bc records of the block. The constraint $n_j \neq 0$ assures that the r_i records are selected from b_i blocks exactly and not less than b_i. The condition $\Sigma n_j = r_i$ ensures that all the possible selections of the r_i records will be selected out. The summation of the products multiplied by the number of ways that b_i blocks may be selected out of the B file blocks, which is $\text{comb}(B, b_i)$, gives the total number of ways that the b_i blocks with r_i records may be selected out of the B file blocks. This number is divided by the number of possible selection of the r_i records out of R file records. The probability distribution follows.

It is clear that the following relations hold:

$$\sum_{b_i=1}^{r_i} \text{P}(b_i) = 1$$

and

$$\sum_{b_i=1}^{r_i} b_i \text{P}(b_i) = B * \left[1 - \frac{\text{comb}(R - bc, r_i)}{\text{comb}(R, r_i)}\right].$$

Consider again that each simple query retrieves b_i blocks. Some of the b_1 and b_2 blocks may be identical. In the sequel by v_2 we denote the number of blocks selected out of the b_2 blocks which do not overlap with some of the b_1 records. Apparently v_2 may range from 0 to b_2. The expected number of block accesses to satisfy the two simple queries is given by the following relation:

$$cost_2 = \frac{1}{\text{comb}(B, b_2)} \sum_{v_2=0}^{b_2} (b_1 + v_2) \times comb(B - b_1, v_2)\,\text{comb}(b_1, b_2 - v_2).$$

The combinations of the summation give the number of ways that the v_2 out of the b_2 blocks do not hit blocks selected by the first query, while $b_2 - v_2$ blocks coincide with some of the b_1 blocks. The corresponding cost is $(b_1 + v_2)$ block accesses. Finally, the denominator gives the number of ways that the b_2 blocks may be selected out of the B file blocks.

Therefore, the total cost for answering a logical query decomposed in two simple queries is:

$$Totcost_2 = \sum_{b_1=1}^{r_1} \sum_{b_2=1}^{r_2} \text{P}(b_1)\,\text{P}(b_2)\, cost_2.$$

If the logical query may be decomposed in three simple queries then the total cost is:

$$Totcost_3 = \sum_{b_1=1}^{r_1} \sum_{b_2=1}^{r_2} \sum_{b_3=1}^{r_3} \text{P}(b_1)\,\text{P}(b_2)\,\text{P}(b_3)\, cost_3,$$

where a more complicated approach is necessary for deriving a new formula for $cost_3$. This cost is:

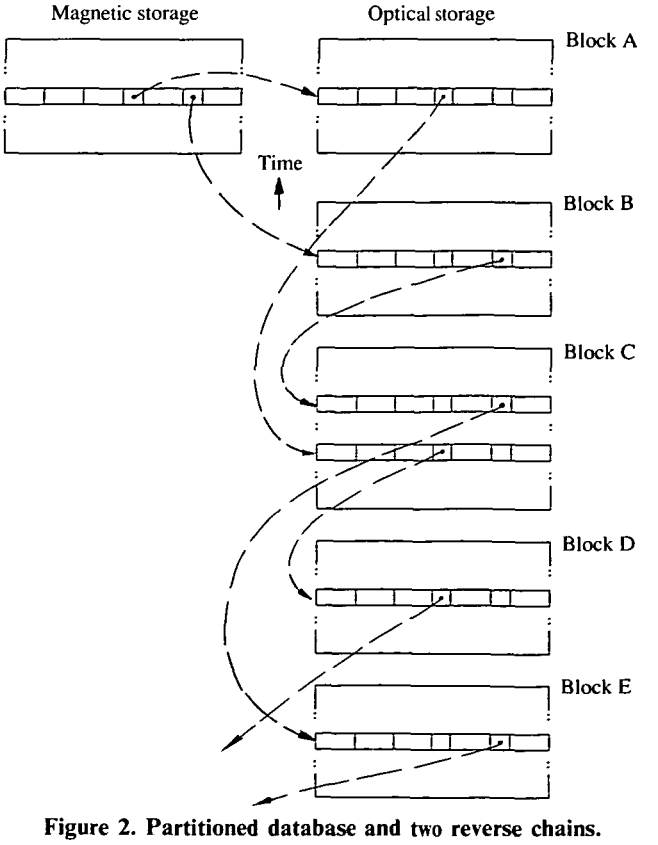

Figure 2. Partitioned database and two reverse chains.

$$cost_3 = \frac{1}{\text{comb}(B,b_2)\text{comb}(B,b_3)} \sum_{v_2=0}^{b_2}$$
$$\times \sum_{v_3=0}^{b_3} (b_1+v_2+v_3)\,\text{comb}(B-b_1,v_2)$$
$$\times \text{comb}(b_1,b_2-v_2)\text{comb}(B-b_1-v_2,v_3)$$
$$\times \text{comb}(b_1+v_2,b_3-v_3).$$

After applying the binomial coefficient properties it is derived that:

$$cost_3 = cost_2 + \frac{1}{\text{comb}(B,b_2)} \sum_{v_2=0}^{b_2} \sum_{v_3=0}^{b_3}$$
$$\times v_3\,\text{comb}(B-b_1-v_2,v_3)$$
$$\times \text{comb}(b_1+v_2,b_3-v_3).$$

Therefore, finally the total cost for a logical query decomposable in n simple ones is:

$$Totcost_n = \sum_{b_1=1}^{r_1} \cdots \sum_{b_n=1}^{r_n} \prod_{i=1}^{n} \mathrm{P}(b_i)\,cost_n,$$

where $cost_n$ is a recursive function of the form:

$$cost_n = cost_{n-1} + \frac{1}{\text{comb}(B,b_n)} \sum_{v_2=0}^{b_2} \cdots \sum_{v_n=0}^{b_n}$$
$$\times v_n\,\text{comb}(B-f_n-1,v_n)\,\text{comb}(f_n-1,v_n).$$

The quantity f_{n-1} gives the number of blocks which have been already accessed by the previous $(n-1)$ queries and is defined as:

$$f_n = b_1 + \sum_{i=2}^{n} v_i.$$

3. Numerical results

The following tables depict some simple numerical results. It is assumed that $R = 100$ records and $B = 20$ blocks; therefore, the block capacity is $bc = 5$. Cost A is the mean cost for answering the two queries independently of each other, while Cost B is the mean cost if the two queries are answered by using the method described in the second section. Table 1 concerns two logical user queries. As expected, greater are the query sizes greater is the gain. Table 2 gives a simple example to show that, as expected again, the gain is greater for greater number of logical queries.

Acknowledgement

Many thanks are due to Antonis and Christos Aletras for their help in experimentation.

Y. MANOLOPOULOS

Division of Electronics and Computer Eng., Department of Electrical Eng., Aristotelian University of Thessaloniki, 54006 Thessaloniki, Greece

References

1. I. Ahn and R. Snodgrass, *Partitioned Storage for Temporal Databases*, *Information Systems*, Vol. 13, No. 4, pp. 369–391 (1988).
2. B. G. Claybrook, *File Management Techniques*, John Wiley (1983).
3. S. B. Yao, Approximating Block Accesses in Database Organizations, *Communications of ACM*, **20**, (4), 260–261 (1977).

Book Review

JAMES W. HOOPER and ROWENA O. CHESTER, *Software Reuse Guidelines and Methods*, Plenum Press, New York and London, 1991 $45.00. ISBN 0-306-43918-2

The idea of reusing software has its origins in the very earliest days of computing when subroutine libraries were first introduced, but its conscious development as a technique with the potential for substantially reducing software development costs is much more recent. Although the concept of reducing costs through extensive software reuse seems at first to be simple and obvious enough, formidable practical obstacles have to be surmounted. These range from purely technical issues such as ways of generalising subroutine interfaces to significant managerial and organisational issues such as the best way of organising a large software production operation to take maximum advantage of any potential for reuse.

These issues have been addressed in a large number of research projects addressing different aspects of the problem of encouraging the introduction of widespread software reuse, but reuse involves so many widely different issues that it is often hard to interrelate the different aspects of the overall problem. With the present book, a general picture of the whole field emerges, and it is hard to overestimate the importance of this for further progress in making widespread reuse a reality. The book arises from a research project sponsored by the US Army: it comprises a very detailed and well-structured commentary on the technical literature on reuse, with accounts of case studies and recommended guidelines for future practice added at appropriate points.

After an introductory chapter, separate chapters deal with managerial and with technical guidelines for the introduction of software reuse, concluding with a brief chapter on 'Getting Started'. The accounts of relevant technical literature are very perceptively written, and many sections conclude with recommended guidelines for future practice – all the guidelines are collected together in a Appendix, with guidelines for reusable Ada code given separately. An extremely comprehensive range of references is given: somewhat confusingly references are given both at the ends of the chapters and in an integrated bibliography at the end of the book.

Anyone who thinks seriously about the problems of software reuse will realise that this is a complicated problem involving both technical and managerial issues, and that different contributions to the subject are often hard to relate even to one another. With this book, there is at last a unifying overview of the whole range of problems relating to software reuse that are carefully and perceptively related to each other and distilled into a series of detailed guidelines for future practice. As an overall tutorial introduction to software reuse and its problems coupled with its guidelines for future practice, the book deserves a wide readership.

PETER WALLIS
Bath

Announcement

1–5 MARCH 1993

CAIA-93, the Ninth IEEE Conference on Artificial Intelligence for Applications, Disneyworld Hilton – Orlando, Florida

The conference is devoted to advancing the application of artificial intelligence techniques to real world problems. This conference provides a forum for synergy between applications and AI techniques. Emphasis at this year's conference will be on new AI paradigms that can have or have had an impact on applications.

Partial list of invited speakers: Patrick Winston, MIT and Ascent Technology, 'Learning and Database Mining'; Wendy Lehnert, University of Massachusetts, 'What we've learned from the DARPA Natural Language Initiative'.

- 1–2 March 1993: Conference tutorial programme
- 3–5 March 1993: Conference technical programme

For registration and additional conference information, contact:

CAIA-93, IEEE Computer Society, 1730 Massachusetts Avenue, NW, Washington, DC 20036-1903. Phone: 202-371-1013.