HOW TO STORE A TRIANGULAR MATRIX

Andrea S. LaPaugh
Richard J. Lipton
Jonathan S. Sandberg

CS-TR-240-89

December 1989

How to Store a Triangular Matrix *

Andrea S. LaPaugh
Richard J. Lipton
Jonathan S. Sandberg
Computer Science Department
Princeton University

Abstract

We consider the problem of storing a triangular matrix so that each row and column is stored as a "vector", i.e the locations form an arithmetic progression. Storing rows and columns as vectors can speed up access significantly. We show that there is no such storage method that does not waste approximately one half of the computer memory.

1 Introduction

As part of our Massive Memory Machine Project [2] we are studying computations that require large data structures. This work has led us to consider the following problem:

Problem: For large n, what is a good method to store an $n \times n$ upper triangular matrix in main memory?

Many computations access a matrix by accessing elements in a row or a column consecutively. Therefore, to serve our needs, a good storage method should provide fast access to both consecutive elements in a row and consecutive elements in a column. We require our storage methods to have the following property:

Stride Property For each row (resp. column) there is an initial address l and a stride s, both depending on the row (resp. column), so that the row (resp. column) elements are stored at locations $l, l+s, l+2s, \ldots$.

A storage method with the stride property allows fast array references. It greatly simplifies the computation needed to get the address of each array element. On machines that support either vector processing or that have multiple memory banks, data can be accessed in parallel [1]. Since main memory is often a major bottleneck such parallel access to memory is very important. On vector machines the cost of reading or writing elements from main memory is very sensitive to their location. If they are "randomly" located, then an expensive memory access is required: this is often called a *gather-scatter*

*This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and by the Office of Naval Research under Contracts Nos. N00014-85-C-0456 and N00014-85-K-0465, and by the National Science Foundation under Cooperative Agreement No. DCR-8420948. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

operation. However, if they are located at $l, l+s, l+2s, \ldots$ then a "block" operation can be performed. On many machines this is an order-of-magnitude faster.

Are there good methods, with the stride property, to store the $n \times n$ triangular matrix *efficiently*? As usual we assume that all memory is allocated as contiguous blocks. So we measure the efficiency of a storage mechanism by the amount of "wastage", i.e. by the number of allocated locations that are not used. Of course, it is trivial to store the triangular matrix so that it has the stride property: just store it in the usual row major order. Then each row has a stride of one and each column has a stride of length n. Storing an triangular matrix, containing $n(n+1)/2$ values, in row major order leaves almost half of the allocated memory unused, and thus is quite inefficient. (The same holds, of course, for column major.)

At best, one can, save a factor of about two in memory;however, such a savings can be very significant. Improvements in memory requirements can have *non-linear* effects: solving a problem with half as much memory has a large payoff. A problem that requires more main memory than is available may not even run, or may run much slower due to excessive disk references in a virtual memory system.

A nice example of this effect is observed in computing the "secondary structure" of certain sequences of *RNA* [4,5,8,9,10]. (This is a problem we are studying, with Doug Welsh, a molecular biologist, as part of our Massive Memory Machine Project.) Secondary structure is determined by a matching of the *RNA* with itself that minimizes its energy. These matchings are found by dynamic programming on large matrices. In particular, they require storing $n \times n$ upper triangular matrices, where n is determined by the length of a strand of *RNA* ($n \approx 3 \times 10^4$ is possible). Molecular biologists find that the size of the triangular array and the high number of resulting page faults is the limiting factor in running their algorithms. For example, runs on much smaller problems ($n \approx 3,000$) have been reported to take months of computer time on *VAX 11/780's* due to paging [8]. Thus, these triangular arrays must fit in random access memory to avoid such "thrashing" i.e. excessive disk references.

We have obtained both upper and lower bounds on the amount of memory required to store an $n \times n$ triangular array so that it has the stride property. The usual row major method of storing a triangular array is optimal, (i.e. it is not possible to waste less than half of the allocated memory). While intuitively this result is clear, the proof requires careful analysis of the possible storage mechanisms. Effectively, the lower bound is a result from diophantine analysis that may be of independent interest.

2 A Lower Bound on Storage

We present an analysis of storage methods that satisfy the stride property. Recall that we allow a distinct arithmetic progression for each row and column. Thus there may be as many as $2n$ distinct progressions used to access the rows and columns of the matrix. For sufficiently large n, we will prove that all such strategies waste space at least $n^2/2 - O(n^{4/3})$.

Our matrices are indexed with x increasing along the horizontal axis, from 0 to $n-1$, and y increasing along the vertical axis from 0 to $n-1$. We choose the lower left corner to be the origin, and since we consider upper triangular matrices, $x \leq y$.

2.1 Definitions

Our first task is to formalize what it means to store a triangular matrix so that it has the stride property.

Definition: A *storage function* for an $n \times n$ triangular matrix is a injective function $f(x,y)$ from $0 \leq x \leq y < n$ to the integers.

A storage function maps the elements of the $n \times n$ triangular array into the integers. It must be injective, i.e. not map two elements of the triangular array to the same location; otherwise, it would not correspond to a valid storage scheme. Since memory is allocated in consecutive blocks, the "span" of the storage function determines the amount of storage (or memory) that must be allocated to the matrix (including wasted storage).

Definition: The storage required by a storage function f (denoted $||f||$) is defined to be

$$\max_{x,y,u,v} |f(x,y) - f(u,v)| + 1.$$

Definition: A storage function f has the *stride property* if for each x in the interval $[0,n-2]$, there is an integer c_x such that for all y, $f(x,y+1) = f(x,y) + c_x$, and for each y in the interval $[1,n-2]$, there is an integer r_y such that for all x, $f(x+1,y) = f(x,y) + r_y$. A storage function which has the stride property is an *arithmetic storage function.*

Our analysis proceeds by transforming the general problem of finding good arithmetic storage functions to a specific question about the diophantine equation $ax+by+cxy$. We show that up to translation, any arithmetic storage function must be of this form. This diophantine form shows that not only must each row and column be accessed via an arithmetic progression, but that the row and column strides are also in an arithmetic progression, each defined by a single constant c. The stride used for the i^{th} row is $a+ci$ and the stride used for the j^{th} column is $b+cj$. Lemma 2.1 states this formally.

Lemma 2.1 *If f is an arithmetic storage function, then there are integers a, b, c, and d such that*

$$f(x,y) = ax + by + cxy + d$$

.

Proof: Let f be an arithmetic storage function. Then f has the stride property. Examine the two paths from (x,y) to $(x+1,y+1)$ for an arbitrary x and y with $x < y$. By the stride property, it follows that

$$r_{y+1} - r_y = c_{x+1} - c_x. \tag{1}$$

Define $\alpha = r_2 - r_1$. Thus, since x and y are arbitrary in equation (1), it follows that for any $y > 0$, $\alpha = r_{y+1} - r_y$. A simple induction shows that for some β and all y, $r_y = \alpha y + \beta$. Now by repeated use of the stride property and this last observation,

$$\begin{aligned} f(x,y) &= f(x-1,y) + r_y \\ &= f(x-2,y) + 2r_y \\ &= f(0,y) + xr_y \\ &= f(0,y) + x(\alpha y + \beta). \end{aligned}$$

But the stride property shows easily that $f(0,y)$ is a linear function of y; hence, $f(x,y)$ has the required form. □

2.2 Main Results

The fundamental problem we are interested is the following: for a given n, how small can $||f||$ be for an arithmetic storage function? Again the row major method yields $||f||$ of approximately n^2. Our main result is that this is best possible, i.e. approximately half the memory must be wasted. The key is that lemma 2.1 proves that this question can be restated as a pure diophantine question:

Problem: For a given n, and constants a, b, c, and d, show that if $f(x,y) = ax + by + cxy + d$ is an arithmetic storage function, then $||f||$ must be at least n^2 up to lower order terms.

Clearly, we can assume that d is zero and that $c \geq 0$. Our analysis of this problem divides into two cases based on whether c is zero or not.

Theorem 2.1 *If $c = 0$, then any arithmetic storage function f has $||f||$ at least $n(n-1)+1$.*

Thus, this theorem proves that the usual row major storage method is optimal, in this case.

Proof: Since c is zero, $f(x,y) = ax + by$. Assume that $||f||$ is less than $n(n-1)+1$. Then we will show that f is not injective; this will prove the theorem. We can assume without loss of generality that $b \geq 0$. Also since f is injective, both a and b must be non-zero. Now since $||f|| \leq n(n-1)$ it follows that

$$f(x,y) - f(u,v) < n(n-1)$$

for any legal values x, y, u, v. We then use this observation as follows:

1. apply it to $f(0, n-1)$ and $f(0,0)$ to show that $b < n$;
2. apply it to $f(n-1, n-1)$ and $f(0,0)$ to show that $a + b < n$;
3. apply it to $f(0, n-1)$ and $f(n-1, n-1)$ to show that $-n < a$.

Now there are two cases: First, assume that $a < 0$. In this case both $f(b, n-1)$ and $f(0, n-1+a)$ are well defined. However, both are equal to $b(n-1+a)$. This contradicts the fact that f is injective. Thus, it must be the case that $a > 0$. Now consider $f(b, n-a-1)$ and $f(0, n-1)$. Both these are well defined: the first since $a + b < n$. But again this contradicts the fact that f is injective since both these have the common value $b(n-1)$. □

Theorem 2.2 *If $c > 0$, then any arithmetic storage function f has $||f||$ at least $n^2 - O(n^{4/3})$.*

Thus, this theorem shows that for c non-zero, approximately the same amount of storage is wasted, i.e. at least $n^2/2$ storage, as in the case when c is zero. In order to prove this theorem we first require a number of lemmas.

Lemma 2.2 *Suppose that $r > 0, s > 0, \alpha, \beta$ are integers, and let A be the following set:*

$$\{(x,y) | 0 \leq x \leq y < n \text{ and } x + \alpha \equiv 0 \bmod r \text{ and } y + \beta \equiv 0 \bmod s\}.$$

Then, $\frac{n^2}{2rs} - O(n/r) - O(n/s) \leq |A| \leq \frac{n^2}{2rs} + O(n/r) + O(n/s)$.

Proof: We can assume that $0 \leq \alpha < r$ and $0 \leq \beta < s$ without loss of generality. Now x must equal $rk - \alpha$ for $k = 1, \ldots, \lfloor (n+\alpha)/r \rfloor$; also y must equal $sl - \beta$ for some $l = 1, \ldots, \lfloor (n+\beta)/s \rfloor$. Then,

$$|A| = \sum_{k=1}^{\lfloor (n+\alpha)/r \rfloor} \sum_{rk - \alpha \leq sl - \beta}^{\lfloor (n+\beta)/s \rfloor} 1 \tag{2}$$

$$= \sum_{k=1}^{\lfloor (n+\alpha)/r \rfloor} \sum_{l = \lceil (rk+\beta-\alpha)/s \rceil}^{\lfloor (n+\beta)/s \rfloor} 1 \tag{3}$$

where we define the sum of 1 from i to j to be 0 whenever $i > j$.

If $r \geq n$, then there is a unique value of x equivalent to α and k only takes on the value 1. Then $|A| = max\{0, \lfloor (n+\beta)/s \rfloor - \lceil (x+\beta)/s \rceil\}$ which clearly satisfies the lemma.

For $r < n$, we will use the fact that $O(r/s) = O(n/s)$. We first derive the upper bound on $|A|$:

$$|A| \leq \sum_{k=1}^{\lfloor (n+\alpha)/r \rfloor} \sum_{l=\lceil (rk+\beta-\alpha)/s \rceil}^{\lceil (n+\beta)/s \rceil} 1$$

Then for each k there is at least one l for which $rk - \alpha \leq sl - \beta$ since

$$\frac{1}{s}(rk + \beta - \alpha) \leq \frac{1}{s}(r\lfloor (n+\alpha)/r \rfloor + \beta - \alpha) \leq \frac{1}{s}(n+\beta).$$

We have

$$|A| \leq \sum_{k=1}^{\lfloor (n+\alpha)/r \rfloor} (\lceil (n+\beta)/s \rceil - \lceil (rk+\beta-\alpha)/s \rceil)$$

and since $(\beta - \alpha)/s \geq -r/s$,

$$\begin{aligned}
|A| &\leq \sum_{k=1}^{\lfloor (n+\alpha)/r \rfloor} \left(\frac{n}{s} - \frac{r}{s}k + \frac{r}{s} + O(1) \right) \\
&\leq \lfloor (n+\alpha)/r \rfloor \left(\frac{n}{s} + \frac{r}{s} + O(1) \right) - \frac{r}{2s}(\lfloor (n+\alpha)/r \rfloor + 1)\lfloor (n+\alpha)/r \rfloor \\
&\leq \frac{n^2}{rs} + O(n/s) + O(n/r) - \frac{r}{2s}\left(\frac{n}{r}\right)\left(\frac{n}{r} - 1\right) \\
&\leq \frac{n^2}{2rs} + O(n/s) + O(n/r).
\end{aligned}$$

We now derive the lower bound on $|A|$. Beginning with equation (3) and noting that $(\beta - \alpha)/s \leq 1$, we have:

$$\begin{aligned}
|A| &\geq \sum_{k=1}^{\lfloor (n+\alpha)/r \rfloor} (\lfloor (n+\beta)/s \rfloor - \lceil (rk+\beta-\alpha)/s \rceil) \\
&\geq \sum_{k=1}^{\lfloor (n+\alpha)/r \rfloor} \left(n/s - \frac{r}{s}k - O(1) \right) \\
&\geq \lfloor (n+\alpha)/r \rfloor \left(\frac{n}{s} - O(1) \right) - \frac{r}{2s}(\lfloor (n+\alpha)/r \rfloor + 1)\lfloor (n+\alpha)/r \rfloor \\
&\geq \frac{n^2}{rs} - O(n/s) - O(n/r) - \frac{r}{2s}\left(\frac{n}{r} + 1\right)\left(\frac{n}{r}\right) \\
&\geq \frac{n^2}{2rs} - O(n/s) - O(n/r).
\end{aligned}$$

□

Lemma 2.3 *Suppose that p, q are distinct primes and α, β are integers, and let A be the following set:*

$$\{(x,y) | 0 \leq x \leq y < n \text{ and } (x+\alpha)(y+\beta) \equiv 0 \bmod pq\}$$

Then, $|A| \geq \frac{n^2}{pq}(1 - \frac{1}{2pq}) - O(n/p + n/q)$.

Proof: The key to the proof is to define the following two sets: let B_1 be the set

$$\{(x,y)|0 \leq x \leq y < n \text{ and } x+\alpha \equiv 0 \bmod p \text{ and } y+\beta \equiv 0 \bmod q\}$$

and let B_2 be the set

$$\{(x,y)|0 \leq x \leq y < n \text{ and } x+\alpha \equiv 0 \bmod q \text{ and } y+\beta \equiv 0 \bmod p\}$$

(Note, these sets are the same except that the roles of p and q have been reversed.) Also let $C = B_1 \cap B_2$. Now B_1 and B_2 are both subsets of A; hence,

$$|A| \geq |B_1| + |B_2| - |C|. \tag{4}$$

By lemma 2.2, both $|B_1|$ and $|B_2|$ bounded below by

$$\frac{n^2}{2pq} - O(n/p + n/q).$$

Now (x,y) is in C provided $0 \leq x \leq y < n$ and $x+\alpha \equiv 0 \bmod pq$ and $y+\beta \equiv 0 \bmod pq$. Thus, again by lemma 2.2, $|C|$ is bounded above by

$$\frac{n^2}{2p^2q^2} + O(n/pq).$$

Equation (4) and the expressions for B_1, B_2, C imply that $|A|$ is at least

$$\frac{n^2}{pq} - \frac{n^2}{2p^2q^2} - O(n/p + n/q).$$

This concludes the proof of the lemma. □

Lemma 2.4 *Suppose that I is an subinterval of the integers. Also assume that there are k elements $x_1, \ldots, x_k$ in I so that for each x_i, $x_i \equiv 0 \bmod m$. Then, $|I| \geq (k-1)m$.*

Proof: Let I be the interval $[\alpha, \beta]$. Clearly, without loss of generality, we can assume that

$$\alpha \leq x_1 < \cdots < x_k \leq \beta.$$

Now it is easy to see that $x_k \geq x_1 + (k-1)m$. Thus, $\beta \geq \alpha + (k-1)m$. Therefore, it follows that $\beta - \alpha \geq (k-1)m$. Since $|I|$ is at least $\beta - \alpha$ the lemma is proved. □

We are now ready to prove theorem 2.2.

Proof of Theorem 2.2: Now $f(x,y)$ is equal to $ax+by+cxy$ with $c > 0$. Choose two distinct primes p and q so that they are approximately equal to $n^{1/3}$. By Bertrand's Postulate [3], we can always do this. Since c is fixed, it is relatively prime to both p and q. Let $0 < d < pq$ be choosen so that $cd \equiv 1 \bmod pq$. Clearly, $g(x,y) = f(x,y) + abd$ is an arithmetic storage function with the same span as f. Therefore, we will prove that g has a large span. A simple calculation shows that $g(x,y) \equiv (cx+b)(y+ad) \bmod pq$. Lemma 2.3, applied to g, therefore, shows that $g(x,y)$ is congruent to 0 modulo pq for at least

$$\frac{n^2}{pq}(1 - \frac{1}{2pq}) - O(n/p + n/q)$$

pairs (x,y). Since g is injective this implies that the span of g includes at least this many values that are congruent to 0 modulo pq. Therefore, lemma 2.4 can be applied to show that the span is at least

$$pq\left(\frac{n^2}{pq}(1 - \frac{1}{2pq}) - O(n/p + n/q)\right)$$

in size. But, this implies that the span is $n^2 - O(n^{4/3})$ and it completes the proof of the theorem. □

3 Conclusion

Using arithmetic progressions in the storage of triangular matrices wastes 50% of the allocated memory. We have proved this by reducing it to a diophantine problem that may be of independent interest.

A generalization of the diophantine problem asks:

Problem: For a given a, b, c, d, q and n, what is $||f||$ if $f(x,y) = (ax + by + cxy + d)_{mod\ q}$ is a injective function from $0 \leq x \leq y < n$ to the integers.

It is interesting that $||f||$ can be much smaller when modular arithmetic is used. In fact, setting $a = \lfloor \frac{n}{2} \rfloor$, $b = \lfloor \frac{n}{2} \rfloor + 1$, $c = 0$ and $q = a(n+b)+b$ gives an f with $||f|| \leq \frac{3}{4}n^2 + O(n)$ for $n \geq 3$. It is also natural to ask similar questions for the full nxn matrix. We can show that, up to symmetry, row major order is the only arithmetic storage scheme on the full matrix which uses no more than n^2 storage.

At a more practical level, for an interleaved memory system to take advantage of arithmetic progressions in accessing rows and columns of a matrix, the progressions must produce memory references which are distributed evenly over the memory banks [7]. For example, consider two interleaved memory banks storing words with odd and even addresses, respectively. The stride used in accessing a row or column element must be odd so that references will alternate between memory banks. In this case, the choice of a and b is very important. Therefore, the characterization of the set of a,b pairs for which storage functions use $n(n-1)+1$ memory locations is an interesting open problem.

References

[1] P. Budnick, D.J. Kuck, "The Organization and use of Parallel Memories," *IEEE Transactions on Computers*, vol. c-20, no. 12, Dec 1971, pp. 1566-1569.

[2] H. Garcia-Molina, R. Lipton and J. Valdes, "A Massive Memory Machine," *IEEE Transactions on Computers*, vol. C-33, no. 5, May 1984, pp. 391-399.

[3] G.H. Hardy and E. M. Wright, "An Introduction to the Theory of Numbers," Oxford University Press, London, 1975.

[4] A.B. Jacobson, L. Good, J. Simonetti, and M. Zucker, "Some Simple Computational Methods to Improve the Folding of Large RNA's," *Nucleic Acids Research*, vol. 12, no. 1, 1984, pp. 45-52.

[5] M. Kanehisa and C. DeLisi, "The Prediction of a Protein and Nucleic Acid Structure: Problems and Prospects," *Acta Applicande Mathematicae*, no. 4, 1985,pp.115-137.

[6] D.E. Knuth, "The Art of Computer Programming," vol. 1, Addison-Wesley, Reading, Ma., 1973.

[7] D.H. Lawrie, "The Prime Memory System for Array Access," *IEEE Transactions on Computers*, vol c-31, no. 5, 1982, pp. 134-141.

[8] R. Nussinov, G. Pieczenik, J.R. Griggs, and D.J. Kleitman, "Algorithms for Looping Matching," *SIAM Journal of Applied Mathematics*, vol. 35, no. 1, 1978, pp. 68-82.

[9] D. Sankoff, J.B. Kruskal, S. Mainville, and R.J. Cedergren, "Fast Algorithms to Determine RNA Secondary Structures Containing Multiple Loops," in *Time Wraps, String Edits, and Macromolecules: Theory and Practice of Sequence Comparisons*, D. Sankoff and J.B. Kruskal editors, Addison-Wesely: Reading, MA, 1983, pp. 93-120.

[10] M. Zucker and P. Stiegler, "Optimal Computer Folding of Large RNA Sequences Using Thermodynamics and Auxiliary Information," *Nucleic Acids Research*, vol. 9, no. 1, 1981, pp. 133-148.