Noisy Sort, A Memory-Intensive Sorting Algorithm

Susumu Horiguchi* and Willard L. Miranker
IBM Thomas J Watson Research Center
Yorktown Heights, New York 10598

Dedicated to Alan J. Hoffman on the occasion of his 65th birthday.

Submitted by Richard A. Brualdi

ABSTRACT

A table-lookup technique for sorting is developed. It is a highly parallel method which develops an approximation to the sort through a single access to an associative memory. A postprocessing step is intended to complete the sort. The scheme is most effective for special data classes.

1. INTRODUCTION

Increasing demand for computing power has stimulated construction of parallel computers [1–3]. Sorting is one of the most studied problems in computer science, both practically and theoretically. Various parallel sorting algorithms have been devised which are tailored to one or another parallel computer system. Networks for odd-even sort and bitonic sort were first described by Batcher [4]. Batcher's fundamental ideas have been extended and adapted to a variety of parallel architectures. Stone [5] showed how to implement Batcher's bitonic sort on the perfect shuffle network. Thompson and Kung [6] implemented the odd-even sort and bitonic sort algorithm on a mesh-connected parallel computer. Nassimi and Sahni [7] proposed a parallel sorting algorithm for a perfect shuffle and unshuffle network. Preparata and Vuillemin [8] proposed an alternative interconnection scheme called the cube-connected cycles, and implemented the bitonic sort on it.

*On leave from Tohoku University, Sendai 980, Japan.

655 Avenue of the Americas, New York, NY 10010 0024-3795/89/$3.50

Many researchers have investigated various parallel sorting algorithms suitable for specific parallel architectures. Although such parallel computers might be available in the near future, the growth in demand for sorting continues to outstrip the growth in processor power. It is therefore of interest to develop another approach which transcends specifics of the parallel architecture.

A number of hardware implementations of associative memories have been proposed [9–12]. These memories allow simultaneous comparison of all the stored data with an external datum. The concept of simultaneous transformation of many data by associative means has been attractive in various applications such as image processing and database management. Huge memories have been extremely expensive in past decades. However, recent developments in VLSI technology makes such memories both inexpensive and easy to produce. This situation has stimulated active research in memory-intensive computation. Here, we propose a new sorting scheme based on such a *memory-intensive approach*. We use the paradigm of an associative memory which allows simultaneous reference to all stored data and external multiple data. In particular, reference to an associative memory is tantamount to the performance of certain arithmetic operations on scalars and vectors as basic operations.

Memory-intensive sorting is composed of three steps: preprocessing, table lookup (the memory access), and postprocessing. We consider a framework where a sample of an actual data set to be sorted is approximately represented by a standard or model data set. We then compose a table (preprocessing) which resides in the associative memory, using this model data set. The new sorting scheme, consisting of a table reference, is named noisy sort, because the scheme does not always give a complete sort (without the postprocessing step).

A heuristic view of noisy sort: We place a set of pairs into an associative memory. These are composed of a set of standard (model) unsorted strings and the corresponding set of pointers sorting those strings. A new unsorted string is shown to the associative memory. The memory outputs the best fit to the sort (set of pointers) of that string. The best fit is in the sense of least squares, and it is based on the information already stored. Imagine that the string to be sorted is plotted: x^j versus j, $j = 1, \ldots, N$. View the resulting plot as a pattern. The pattern is recognized by identification of the permutation which orders it. Thus noisy sort is *sorting by pattern recognition*.

Noisy sort is a powerful sorting scheme in special fields of application. We expect that its scope will be considerably widened with further study.

In Section 2, the concept and techniques of noisy sort are presented. For ease of comprehension, simple examples are introduced. Properties of the constructs associated with the technique are developed. The results of simulation for noisy sort are shown in Section 3. The expected accuracy of

noisy sort is discussed for data which are Gaussian distributed about the model data. Conclusions are given in Section 4.

2. NOISY SORT

2.1. *Concept*

Let x be a list of N real numbers to be sorted. Consider x to be a column vector in $\mathbb{R}^N$, and let $\mathrm{CP}(x)$ be the collection of $N!$ vectors, the permutations of x taken in some order, hereafter fixed. $\mathrm{CP}(x)$ will also denote the $N \times N!$ matrix composed of these vectors, in that order, as columns, $\mathrm{CP}(x)$ will be called a CP matrix (of x), that is, a complete permutation matrix (of x). Let $x = (x^1, x^2, \ldots, x^N)^T$ be a sorted list of data, and let $X = \mathrm{CP}(x)$. We call x^j the jth principal component of X. [x^j is the $(j,1)$ entry of X.] Let y be the vector of integers $(1,2,\ldots,N)^T$, and let[1] $Y = \mathrm{CP}(y)$. Then y_i i Y is the destination address set into which x_i in X should be stored by sorting. Sorting is a mapping $\tau: \mathrm{CP}(x) \to \mathrm{CP}(y)$. Equivalently, τ is a mapping of $\{x_i\}_1^{N!}$ onto $\{y_i\}_1^{N!}$,

$$y_i = \tau(x_i), \qquad i = 1,2,\ldots,N!.$$

There exists an $N \times N$ matrix M such that

$$y_i = Mx_i, \qquad i = 1,2,\ldots,N!,$$

in sense of least squares [9]. That is, M solves the problem

$$\min_M \|MX - Y\|^2 = \min_M \mathrm{Tr}(MX - Y)^T(MX - Y) = \min_M \sum_{i=1}^{N!} \|Mx_i - y_i\|^2$$

Here $\|\ \|$ denotes the Euclidean norm in $\mathbb{R}^N$. Let

$$X^+ = X^T(XX^T)^{-1}.$$

X^+ is the pseudoinverse of X. M is given by [9]

$$M = YX^+.$$

[1] Although we shall not make use of it here, there are sorting contexts in which the first column of Y could be an arbitrary permutation of y.

Multiplication by M is an operational representation of table lookup by access to an associative memory [9]. This is a highly parallel operation, since the table lookup occurs in one memory access cycle.

Associative Memory Paradigm Consider the pairs of vectors (y_i, x_i), $i = 1, \ldots, K$. The y_i are p-vectors, and the x_i are q-vectors. Let Y be the $p \times K$ matrix composed of the y_i in order columnwise, and let X be the $q \times K$ matrix corresponding analogously to the x_i. An associative memory is a device into which the pairs

$$(y_i, x_i), \qquad i = 1, \ldots, K,$$

are loaded. The memory is interrogated by an arbitrary q-vector ξ. It produces a p-vector η as output, where $\eta = M\xi$. The $p \times q$ matrix M is YX^+. In the case of sorting, at hand, $p = q = N$ and $K = N!$.

The computation of $M\xi$ has complexity $O(N^2)$. We shall see later that in the case of sorting, at hand, M always has a special structure so that the complexity of $M\xi$ is, in fact, $O(N)$ and even $O(\log N)$ if performed in parallel. However, the use of a special piece of hardware, an associative memory, provides the product $M\xi$ with complexity which is $O(1)$ [9].

We now define the noisy sort of any vector $\xi \in \mathbb{R}^N$. It consists of the application of M to ξ followed by a rounding operation. Let

$$\tilde{\eta} = M\xi.$$

In general $\tilde{\eta} \neq \tau(\xi)$, that is, $\tilde{\eta}$ is not necessarily the sort corresponding to ξ. Indeed $\tilde{\eta}$ is not necessarily even a permutation of integers. Let $\square : \mathbb{R} \to y$ be rounding to nearest integer (with ties resolved by some arbitrarily fixed rule). $\square$ is applied to a vector componentwise. We define

$$\eta = \square M\xi \tag{2.1}$$

to be the noisy sort of ξ. We do not require that the rounding, even the tie resolution, advance the sorting. On the other hand, we don't proscribe it either. η in (2.1) is a preliminary estimation for the sort (the noisy sort). Given a noisy sort, a serial algorithm such as bucket sort should be used as the postprocessing step to obtain the complete sort.

It remains an open question to characterize the complexity of this postprocessing step. We do not expect this complexity to be small in general, but we do expect it to be small in many cases. For the cases we have

experimented with (see Section 3 on simulation), we find that noisy sort is correct except for clusters of unsorted elements. These clusters correspond to the ties, just referred to. The clusters are correctly disposed, but the elements within them are not sorted. When the number of clusters as well as the width of the maximal cluster is small compared to N, the postprocessing step will itself be of small complexity.

2.2. *Simple Example*

A simple example will clarify the mechanics of noisy sort. Let $N = 3$. Then

$$Y = \begin{bmatrix} y^1 & y^1 & y^2 & y^2 & y^3 & y^3 \\ y^2 & y^3 & y^1 & y^3 & y^1 & y^2 \\ y^3 & y^2 & y^3 & y^1 & y^2 & y^1 \end{bmatrix},$$

where $(y^1, y^2, y^3) = (1,2,3)^2$;

$$X = \begin{bmatrix} x^1 & x^1 & x^2 & x^2 & x^3 & x^3 \\ x^2 & x^3 & x^1 & x^3 & x^1 & x^2 \\ x^3 & x^2 & x^3 & x^1 & x^2 & x^1 \end{bmatrix};$$

$$X^+ = \begin{bmatrix} z^1 & z^2 & z^3 \\ z^1 & z^3 & z^2 \\ z^2 & z^1 & z^3 \\ z^2 & z^3 & z^1 \\ z^3 & z^1 & z^2 \\ z^3 & z^2 & z^1 \end{bmatrix}.$$

Here

$$z^j = \frac{s^3 - t^2(2x^j - s)}{6s^2(3t^2 - s^2)}, \qquad j = 1,2,3,$$

[2] We suppose that data are sorted into ascending order; if in descending order, $(y^1, y^2, y^3) = (3,2,1)$. In principal any of the six permutations could be the first column of Y.

where

$$s = x^1 + x^2 + x^3,$$

$$t^2 = (x^1)^2 + (x^2)^2 + (x^3)^2.$$

M is given by

$$M = \begin{bmatrix} a & b & b \\ b & a & b \\ b & b & a \end{bmatrix},$$

where

$$a = 2(y^1 z^1 + y^2 z^2 + y^3 z^3),$$

$$b = y^1(z^2 + z^3) + y^2(z^1 + z^3) + y^3(z^1 + z^2).$$

Consider a numerical example for $N = 3$. Y, X, X^+, and M are given by

$$Y = \begin{bmatrix} 1 & 1 & 2 & 2 & 3 & 3 \\ 2 & 3 & 1 & 3 & 1 & 2 \\ 3 & 2 & 3 & 1 & 2 & 1 \end{bmatrix},$$

$$X = \begin{bmatrix} 0.9067 & 0.9067 & 3.5211 & 3.5211 & 6.1482 & 6.1482 \\ 3.5211 & 6.1482 & 0.9067 & 6.1482 & 0.9067 & 3.5211 \\ 6.1482 & 3.5211 & 6.1482 & 0.9067 & 3.5211 & 0.9067 \end{bmatrix},$$

$$X^+ = \begin{bmatrix} -0.0478 & 0.0157 & 0.0794 \\ -0.0478 & 0.0794 & 0.0157 \\ 0.0157 & -0.0478 & 0.0794 \\ 0.0157 & 0.0794 & -0.0478 \\ 0.0794 & -0.0478 & 0.0157 \\ 0.0794 & 0.0157 & -0.0478 \end{bmatrix},$$

$$M = \begin{bmatrix} 0.4435 & 0.0619 & 0.0619 \\ 0.0619 & 0.4435 & 0.0619 \\ 0.0619 & 0.0619 & 0.4435 \end{bmatrix}.$$

An example of specific data ξ and their corresponding $\tilde{\eta}$ and $\square\tilde{\eta}$ is

$$\xi = \begin{bmatrix} 3.4316 \\ 5.9832 \\ 1.1080 \end{bmatrix},$$

$$\tilde{\eta} = \begin{bmatrix} 1.9609 \\ 2.9346 \\ 1.0743 \end{bmatrix},$$

$$\square\tilde{\eta} = \begin{bmatrix} 2 \\ 3 \\ 1 \end{bmatrix}.$$

In this case the noisy sort is the actual result.

2.3. *Matrix M*

The products of two $N \times N!$ matrices and the inverse of an $N \times N$ matrix are required to calculate M. This is impractical indeed for even moderate values of N. We now show how to calculate M in closed form, avoiding such impractical matrix calculations. This leads to economies in the study and simulation of noisy sort. We expect corresponding economies in hardware implementations as well.

A matrix of the form

$$\begin{bmatrix} a & b & b & \cdots & b & b \\ b & a & b & \cdots & b & b \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ b & b & b & \cdots & a & b \\ b & b & b & \cdots & b & a \end{bmatrix}$$

will be called an (a, b) matrix, and the class of such matrices will be denoted by $\mathcal{M}(a, b)$. We shall show that $M \in \mathcal{M}(a, b)$, and determine a and b in closed form.

The class of CP matrices will be denoted by $\mathcal{M}$(CP). The transpose of a CP matrix will be called a TCP matrix, and the class of TCP matrices will be denoted by $\mathcal{M}$(TCP). Consider the following lemma.

LEMMA 1. *If* $Y \in \mathcal{M}$(CP) *and* $Z \in \mathcal{M}$(TCP) *then* $YZ \in \mathcal{M}(a, b)$.

Proof. As a $(k+1)\times(k+1)!$ CP matrix, Y_{k+1} may be written as follows:

$$Y_{k+1}=\begin{bmatrix} y^1 & \cdots & y^1 & y^2 & \cdots & y^2 & \cdots & y^{k+1} & \cdots & y^{k+1} \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ Y_k^1 & & & Y_k^2 & & \cdots & & Y_k^{k+1} & & \end{bmatrix}, \quad (2.2)$$

where Y_k^j is a $k\times k!$ CP matrix composed of $(y^1,\ldots,y^{j-1},y^{j+1},\ldots,y^{k+1})$. Similarly Z_{k+1}, as a $(k+1)!\times(k+1)$ TCP matrix, may be written as

$$Z_{k+1}=\begin{bmatrix} z^1 & \\ \vdots & Z_k^1 \\ z^1 & \\ z^2 & \\ \vdots & Z_k^2 \\ z^2 & \\ \vdots & \\ z_{k+1} & \\ \vdots & Z_k^{k+1} \\ z_{k+1} & \end{bmatrix}, \quad (2.3)$$

where Z_k^j is a $k!\times k$ TCP matrix composed of $(z^1,\ldots,z^{j-1},z^{j+1},\ldots,z^{k+1})$. Let $W=Y_{k+1}Z_{k+1}$:

$$W=\begin{bmatrix} W_{11} & W_{12} \\ W_{21} & W_{22} \end{bmatrix},$$

where W_{11} is a scalar, W_{12} is a row vector, W_{21} is a column vector, and W_{22} is a $k\times k$ matrix. Let $S_y=\sum_{i=1}^{k+1}y^i$, $S_z=\sum_{i=1}^{k+1}z^i$, and $S_{yz}=\sum_{i=1}^{k+1}y^iz^i$. Then using (2.2) and (2.3) we deduce the form of $W_{11}, W_{12}, W_{21}, W_{22}$. First,

$$W_{11}=k!\sum_{i=1}^{k+1}y^iz^i=k!S_{yz}.$$

Next, each of the k elements of W_{12} is equal to

$$w_{12}=(k-1)!\sum_{i=1}^{k+1}\left[y^i\left(S_z-z^i\right)\right]=(k-1)!\left[S_yS_z-S_{yz}\right].$$

Similarly each of the k elements of W_{21} is equal to

$$w_{21} = (k-1)! \sum_{i=1}^{k+1} \left[\left(S_y - y^i \right) z^i \right] = (k-1)! \left[S_y S_z - S_{yz} \right].$$

Clearly $w_{21} = w_{12}$. Finally

$$W_{22} = \sum_{j=1}^{k+1} Y_k^j Z_k^j.$$

$Y_k^j Z_k^j$ is an (a_k^j, b_k^j) matrix, where

$$a_k^j = (k-1)! \sum_{i=1, i \neq j}^{k+1} y^i z^i = (k-1)! \left[S_{yz} - y^j z^j \right]. \tag{2.4}$$

Then all diagonal elements of W_{22} are equal to a_{k+1} say, where

$$a_{k+1} = \sum_{j=1}^{k+1} a_k^j = \sum_{j=1}^{k+1} (k-1)! \left[S_{yz} - y^j z^j \right] = k! S_{yz}.$$

That is, $a_{k+1} = W_{11}$. Similarly

$$b_k^j = (k-2)! \left[\left(S_z - z^j \right) \left(S_y - y^j \right) - \left(S_{yz} - y^j z^j \right) \right]. \tag{2.5}$$

Then all off-diagonal elements of W_{22} are equal to b_{k+1} say, where

$$b_{k+1} = \sum_{j=1}^{k+1} b_k^j = \sum_{j=1}^{k+1} (k-2)! \left[\left(S_z - z^j \right) \left(S_y - y^j \right) - \left(S_{yz} - y^j z^j \right) \right]$$

$$= (k-1)! \left[S_y S_z - S_{yz} \right].$$

That is, $b_{k+1} = w_{12} = w_{21}$. Then W is an (a_{k+1}, b_{k+1}) matrix. ■

From Lemma 1, we have, for any CP matrix X,

$$XX^T = \begin{bmatrix} q & r & r & \cdots & r \\ r & q & r & \cdots & r \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r & r & \cdots & q & r \\ r & r & \cdots & r & q \end{bmatrix}.$$

Using (2.4) and (2.5), q and r are given by

$$q = (N-1)!t^2 \tag{2.6}$$

$$r = (N-2)!(s^2 - t^2), \tag{2.7}$$

where

$$s = \sum_{j=1}^{N} x^j, \qquad t^2 = \sum_{j=1}^{N} (x^j)^2. \tag{2.8}$$

Now consider the following lemma.

LEMMA 2. *If* $A \in \mathcal{M}(a, b)$, $a \neq b$, $a \neq (1-N)b$ *then* A^{-1} *exists, and* $A^{-1} \in \mathcal{M}(c, d)$, *where*

$$c = \frac{a + (N-2)}{(a-b)[a + (N-1)b]},$$

$$d = -\frac{b}{(a-b)[a + (N-1)b]}.$$

Proof. Let $B \in \mathcal{M}(c, d)$. Then

$$\text{diag}(BA) = \text{diag}(AB) = ac + (N-1)bd = 1,$$

$$\text{off-diag}(BA) = \text{off-diag}(AB) = bc + [a + (N-2)b]d = 0. \qquad \blacksquare$$

Using Lemma 2, we have

$$(XX^T)^{-1} = \begin{bmatrix} c & d & d & \cdots & d \\ d & c & d & \cdots & d \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ d & d & \cdots & c & d \\ d & d & \cdots & d & c \end{bmatrix}. \tag{2.9}$$

Moreover,

$$c = \frac{q + (N-2)r}{(q-r)(q+(N-1)r)}, \tag{2.10}$$

$$d = -\frac{q}{(q-r)(q+(N-1)r)}. \tag{2.11}$$

Inserting (2.6) and (2.7) into (2.10) and (2.11), we get

$$c = \frac{(N-2)s^2 + t^2}{(N-1)!s^2(Nt^2 - s^2)},$$

$$d = \frac{s^2 - t^2}{(N-1)!s^2(Nt^2 - s^2)}.$$

Next consider the following lemma.

LEMMA 3. *If* $W \in \mathcal{M}(\text{TCP})$ *and* $A \in \mathcal{M}(a, b)$, *then* $WA \in \mathcal{M}(\text{TCP})$.

Proof. Let w^j be the jth principal component of the TCP matrix W. Let σ_i denote the ith permutation vector of integers $(1,2,\ldots,N)$. Let σ_i^k be the kth integer in the permutation σ_i. Let $S_w = \sum_{k=1}^{N} w^k$, and let z_{ij} be the (i, j)th element of WA. We have

$$z_{ij} = aw^{\sigma_i^j} + b \sum_{k=1,\, k \neq j}^{N} w^{\sigma_i^k} = (a-b)w^{\sigma_i^j} + bS_w. \tag{2.12}$$

The (i, j)th element of WA is determined only by σ_i^j. This implies $WA \in \mathcal{M}(\text{TCP})$. ∎

Using (2.9) and Lemma 3, we deduce the following corollary.

COROLLARY 4. $X^+ \in \mathcal{M}(\mathrm{TCP})$.

Let z^j be the jth principal component of X^+. From (2.12), we have

$$z^j = cx^j + d(s - x^j).$$

Finally we have

THEOREM 5. $M \in \mathcal{M}(a, b)$, *where*

$$a = (N-1)! \sum_{j=1}^{N} y^j z^j, \tag{2.13a}$$

$$b = (N-2)! \sum_{j=1}^{N} y^j (S_z - z^j). \tag{2.13b}$$

Furthermore

$$a = \frac{(N-1)su - \frac{1}{2}(s^2 - t^2)N(N+1)}{s(Nt^2 - s^2)}, \tag{2.14}$$

$$b = \frac{\frac{1}{2}t^2 N(N+1) - su}{s(Nt^2 - s^2)}, \tag{2.15}$$

where s *and* t *are given in* (2.8) *and*[3]

$$u = \sum_{j=1}^{N} y^j x^j. \tag{2.16}$$

Proof. The proof follows directly from Lemma 1 and Corollary 4. ■

Using these results, we see that the complexity for calculating M is reduced to $O(N)$ (indeed, to $5N$ + constant). We close this section with the following observation.

[3] In the case that data are sorted into ascending order, $y^j = j$ in (2.13a), (2.13b), and (2.16).

LEMMA 6. *If the principal components of X are of the form* $x^j = jh$, $j = 1,2,\ldots,N$, *where h is constant, then* $M \in \mathcal{M}(a,0)$, *that is,* $M = aI$.

Proof. Substituting $x^j = jh$ and $y^j = j$ into (2.8) and (2.16), we have $s = N(N+1)/2$, $t^2 = h^2N(N+1)(2N+1)/6$, and $u = hN(N+1)(2N+1)/6$. Inserting these three relations into (2.15), we get $b = 0$. ■

3. SIMULATION

The scheme proposed here operates on the premise that the data set to be sorted is representable by a model data set. The quality of noisy sort as an initial phase of a sorting procedure depends on the validity of this premise.

Let the components x^j, $j = 1,2,\ldots,N$, of the model data be Gaussian, $N(jh + k, \sigma)$, $j = 1,2,\ldots,N$. Here h and k are specified positive increments. We consider two ways of specifying actual data ξ. In the first case, E1, ξ^j is also $N(jh + k, \sigma)$. In the second case, E2, ξ^j is $N(x^j, \sigma)$. In both cases the quality of noisy sort depends on σ and N.

For each sample of actual data ξ, we execute the noisy sort to produce $\eta = \square M\xi$. With $y = \tau(\xi)$ denoting the correct sort, we determine a corresponding figure of merit Q:

$$Q(\sigma, N) = \frac{1}{N} \sum_{j=1}^{N} \delta(\eta^j - y^j).$$

Here $\delta(\omega)$ is the Kronecker delta. Q is averaged over a number $I_{\max}$ of cases. We call this average $\bar{Q}(\sigma, N)$.

First we show the results for case E1. Figure 1 is a plot of $\bar{Q}(\sigma, N)$ versus N for each of a set of σ. When N is large, $\bar{Q}$ is almost a constant which depends on σ. Figure 2 is a plot of $\bar{Q}$ versus σ for each of a set of N. It is clear that accuracy increases as the standard deviation decreases. This verifies the premise of efficacy of noisy sort in situations where the data to be sorted are well represented by model data.

Next we show the results for case E2. Figure 3 is a plot of $\bar{Q}(\sigma, N)$ versus N for each of a set of σ. For large N, the difference between Figure 1 and Figure 3 is slight. $\bar{Q}(\sigma, N)$ for case E1 majorizes $\bar{Q}(\sigma, N)$ for case E2. Figure 4 is a plot for case E2 of $\bar{Q}$ versus σ for each of a set of N.

Next consider the simulation for an actual data set with Gaussian distribution in which the mean value of actual data ξ^j is equal to the model data x^j ($j = 1,2,\ldots,N$), that is, case E2. Let $e^j = y^j - \tilde{\eta}^j$. Now $\xi^1, \xi^2, \ldots, \xi^N$ are

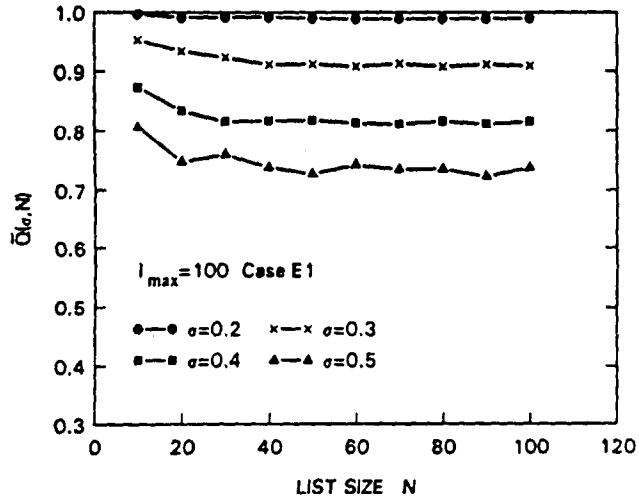

FIG. 1. The average accuracy $\bar{Q}(\sigma, N)$ vs. N for each of a set of σ for case E1.

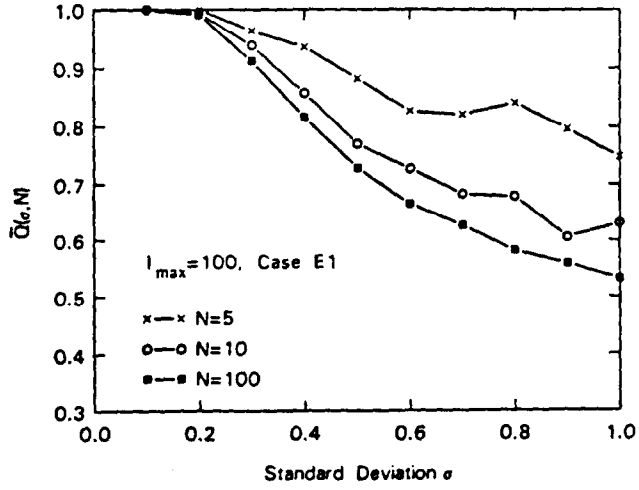

FIG. 2. The average accuracy $\bar{Q}(\sigma, N)$ vs. σ for each of a set of N for case E1.

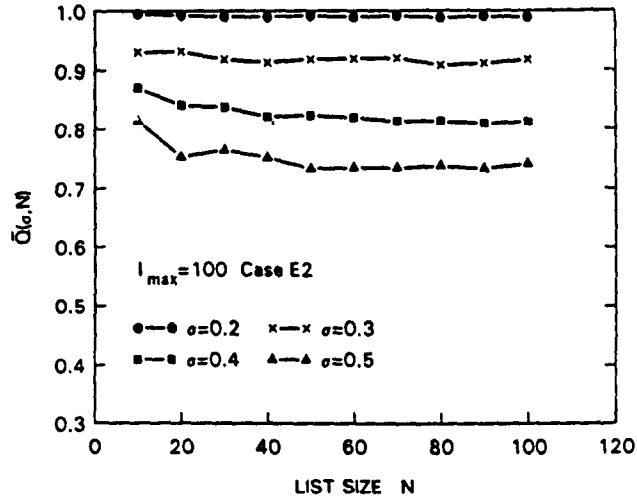

FIG. 3. The average accuracy $\bar{Q}(\sigma, N)$ vs. N for each of a set of σ for case E2.

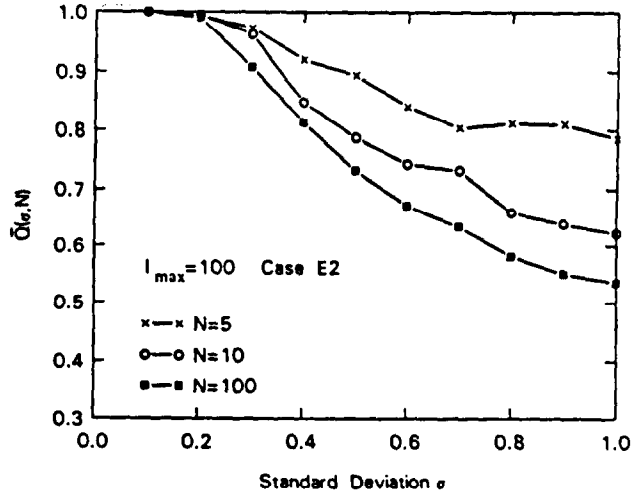

FIG. 4. The average accuracy $\bar{Q}(\sigma, N)$ vs. σ for each of a set of N for case E2.

mutually independent random variables. Then, for the mean value $m(e^j)$ of e^j, we have

$$m(e^j) = y^j - \sum_{k=1}^{N} M_{jk} m(\xi^k) = y^j - \sum_{k=1}^{N} M_{jk} x^k,$$

where M_{jk} is the (j,k)th element of matrix M. For the variance $\sigma^2(e^j)$ we have

$$e^2(e^j) = \sum_{k=1}^{N} M_{jk}^2 \sigma^2(\xi^k) = \sum_{k=1}^{N} M_{jk}^2 \sigma^2 = \left[a^2 + (N-1)b^2\right]\sigma^2, \quad (3.1)$$

since $M \in \mathcal{M}(a,b)$. We denote $\sigma^2(e^j)$ by σ_e^2, since it is independent of j. Noisy sort is correct if

$$e^1 < \tfrac{1}{2}, \qquad |e^j| < \tfrac{1}{2}, \quad j = 2, \ldots, N-1, \quad \text{and} \quad e^N > \tfrac{1}{2}.$$

The probability that noisy sort is correct is

$$P_l \equiv \Pr\left[\left\{e^1 < \tfrac{1}{2}\right\} \bigcup_{j=2}^{N-1} \left\{|e^j| < \tfrac{1}{2}\right\} \cup \left\{e^N > \tfrac{1}{2}\right\}\right].$$

All members of the following relations are lower bounds for P_l:

$$\Pr\left[\bigcup_{j=1}^{N} \left\{|e^j| < \tfrac{1}{2}\right\}\right] = \prod_{j=1}^{N} \Pr\left\{|e^j| < \tfrac{1}{2}\right\}$$

$$= \prod_{j=1}^{N} \left[1 - \Pr\left\{|e^j| \geq \tfrac{1}{2}\right\}\right]$$

$$\geq \prod_{j=1}^{N} \left[1 - \Pr\left\{\left|e^j - m(e^j)\right| \geq \tfrac{1}{2} - \left|m(e^j)\right|\right\}\right]. \quad (3.2)$$

Using the Chebyschev inequality for the rightmost member of (3.2), we obtain a lower estimate for P_l. In particular,

$$P_l \geqslant \prod_{j=1}^{N} \left(1 - \frac{4\sigma_e^2}{\left[1 - 2\left|m(e^j)\right|\right]^2} \right).$$

$m(e^j)$ is zero in the sense of least squares. Let us neglect it and consider the case $x^j = j$. In this case, $a = 1$ and $b = 0$ [cf. (2.14), (2.15)], and $\sigma_e^2 = \sigma^2$ [cf. (3.1)]. Then for small σ we have

$$P_l > (1 - 4\sigma^2)^N \simeq 1 - 4N\sigma^2.$$

4. CONCLUSION

A new sorting scheme for memory-intensive computation has been proposed. The scheme is named noisy sort. Noisy sort is useful for data sets which are represented by a model data set. The expected accuracy of noisy sort has been evaluated through simulation.

Noisy sort is a table-lookup method for sorting. Table lookup or memory access is modeled in terms of a matrix M, which is shown to be a so-called $\mathcal{M}(a, b)$ matrix. Such matrices, depending only on a pair of constants, yield economies in the method and possibly in the corresponding memory architecture as well. Associative memories are examples of a possible such architecture, but the detailed form of such memories more specifically relevant to noisy sort remains to be studied.

For completely general data, there may be no advantage to noisy sort. We anticipate that the scope of noisy sort will be considerably widened with further study.

The authors thank Dr. Craig Douglas and Dr. Andrew Winkler for stimulating discussions on these topics.

REFERENCES

1 W. D. Hillis, *The Connection Machine*, MIT Press, Cambridge, 1985.

2 G. F. Pfister, W. C. Brantley, D. A. George, S. L. Harvey, W. J. Kleinfelder, P. K. McAuliffe, E. A. Melon, V. A. Norton, and J. Weiss, IBM research parallel processor prototype (RP3): Introduction and architecture, in *Proceedings of the* 1985 *International Conference on Parallel Processing*, 1985, pp. 764–771.

3 D. Gajski, D. Kuck, D. Lawrie, and A. Sameh, Cedar—A large scale multiprocessor, in *Proceedings of the International Conference on Parallel Processing*, 1983, pp. 524–529.
4 K. D. Batcher, Sorting networks and their applications, in *Proceedings of AFIPS Conference* 32, 1968, pp. 307–314.
5 H. S. Stone, Parallel processing with the perfect shuffle, *IEEE Trans. Comput.* C-20(2):153–161 (1971).
6 C. D. Thompson and H. T. Kung, Sorting on a mesh-connected parallel computer, *Comm. ACM* 20:151–161 (1971).
7 D. Nassimi and S. Sahni, Bitonic sort on a mesh-connected parallel computer, *IEEE Trans. Comput.* C-28:2–7 (1979).
8 F. P. Preparata and J. Vuillemin, The cube-connected cycles: A versatile network for parallel computation, *Comm. ACM* 24(5):300–309 (1981).
9 T. Kohonen, *Associative Memory: A System-Theoretical Approach*, Springer-Verlag, Berlin, 1977.
10 R. H. Fuller and R. H. Bird, An associative parallel processor with application to picture processing, in *Proceedings of FJCC*, 1965, pp. 105–116.
11 S. S. Yau and H. S. Fung, Associative processor architecture: A survey, in *Proceedings of the* 1975 *Computer Conference on Parallel Processing*, 1975, pp. 1–14.
12 W. A. Davis and D. L. Lee, Fast search algorithm for associative memories, *IEEE Trans. Comput.* C-35(4):456–461 (1986).

Received 7 March 1988; final manuscript accepted 20 September 1988