Parallel Multisearch in Sorted Matrices

Amit Jain
Dept. of Math. & Computer Science
Boise State University
Boise, Idaho 83725
email: amit@cs.idbsu.edu

Abstract

We show that given an $n \times n$ matrix with sorted rows and sorted columns and r elements $z_1 < z_2 < \ldots < z_r$, the problem of multisearch and ranking for these r elements in the matrix can be solved in $O(\log n + \log r)$ time using $O(rn)$ work on the EREW PRAM.

keywords: parallel algorithms, multisearch, sorted matrices

1 Introduction

We propose parallel algorithms for search and multisearch in sorted matrices. Search and selection in sorted matrices has received considerable attention [1, 2, 3, 4, 5, 6, 7]. Searching in sorted matrices also has applications in computational geometry [8], and in multiplying sparse polynomials [7].

An $n \times n$ matrix M is a sorted matrix if each row and column is in nondecreasing order. The search problem is to determine whether a given element z occurs in M. A related problem is that of ranking the element z, which is to find the number of elements in M that are $\leq z$. Sequentially, the problem can be solved optimally in $O(n)$ time [7, 2]. The multisearch problem is to search for r elements $z_1 < z_2 < \ldots < z_r$ in an $n \times n$ sorted matrix.

We will use the EREW PRAM model to design parallel algorithms. We also use the Work-Time presentation style for parallel algorithms. For details on these, see [9].

2 Parallel Search

The sequential algorithm for search appears in Figure 1. The algorithm is easily modified to compute the rank as well. This algorithm is not easily parallelizable. Sorted matrices satisfy the property of being totally monotone. Parallel search algorithms for such matrices [10, 11] are inefficient if used for searching in sorted matrices since they require $O(n \log n)$-work and are also much more complicated than what we propose.

Algorithm 1: search(z,M)
    $\{M[i,j] \leq M[i,j+1],\ 1 \leq i < n$ sorted rows
    $M[i,j] \leq M[i+1,j],\ 1 \leq j < n$ sorted columns $\}$

    $i \leftarrow n,\ j \leftarrow 1$
    while $((\ i \geq 1)$ **and** $(j \leq n))$ **do**
        if $(M[i,j] = z)$ **then**
            return (i,j)
        if $(z < M[i,j])$ **then**
            $i \leftarrow i - 1$
        else
            $j \leftarrow j + 1$
    endwhile
    return *unsuccessful search*
end

Figure 1: Sequential search in a sorted matrix.

Sarnath and He [12] have proposed an optimal parallel algorithm on CREW PRAM for searching in a sorted matrix requiring $O(\log \log n)$ time and $O(n)$ work. Their algorithm is based upon the technique of successive refinement. A straightforward adaptation of their algorithm to the EREW PRAM takes $O(\log n \log \log n)$ time, which is not optimal. We modify their algorithm so it runs optimally in $O(\log n)$ time and $O(n)$ work on the EREW PRAM. Our algorithm uses the technique of *partitioning* the matrix into many small submatrices, eliminating all but $\Theta(n \log n)$ elements of the matrix M in the process. See Figure 2. For correctness and runtime analysis, see [13]. Consequently,

Theorem 2.1 *Given an $n \times n$ sorted matrix M and an element z, the problem of searching and ranking for z in the matrix M as well as ranking can be solved in $O(\log n)$ time using $O(n)$ work on the EREW PRAM.*

Algorithm 3: Parallel_Search

Step 1 For each column with index $c_i = i \times \log n$, where $i = 1, 2, \ldots t (= n/\log n)$, search the column c_i for a row index p_i such that $M[p_i, c_i] < z < M[p_i + 1, c_i]$. (If z is equal to either of these elements the algorithm terminates.) For each i, $1 \le i \le t$, retain the elements within the submatrix $M[p_i + 1, p_{i-1} : c_{i-1}, c_i]$ and discard the rest.

Step 2 For each row with index $r_i = i \times \log n$, where $i = 1, 2, \ldots t (= n/\log n)$, search the retained elements in the row r_i for a column index q_i such that $M[r_i, q_i] < z < M[r_i, q_i + 1]$. (If z is equal to either of these elements the algorithm terminates.) For each i, $1 \le i \le t$, retain the elements within the submatrix $M[r_{i-1}, r_i : q_i + 1, q_{i-1}]$ and discard the rest.

Step 3 Assign one processor to each remaining submatrix and use the optimal sequential algorithm.

end

Figure 2: Parallel algorithm for search in a sorted matrix.

3 Parallel Multisearch

In the multisearch problem we search for r elements $z_1 < z_2 < \ldots < z_r$ in an $n \times n$ sorted matrix on the EREW PRAM. If the search-elements are unsorted, they can be sorted in $O(\log r)$ time using r processors [14].

3.1 A Simple Pipelined Algorithm

In each step of the sequential algorithm, the search either moves up one row or right one column. Let $P_1, P_2, \ldots, P_r$ be the r processors, each processor associated with searching for one element. Then the path traced by processor P_1 while searching for z_1 is bounded below by the paths traced by processors searching for larger elements. However, segments of the step-shaped boundary may be common to two or more searches. For more details, see [13]. Delaying the start of each search by one step ensures there are no read conflicts. Extending this argument yields an $O(n + r)$-time parallel algorithm for r searches on the EREW PRAM.

3.2 The Main Parallel Algorithm

We use a result on multiple binary search as a subroutine.

Theorem 3.1 (Chen [15]) *Given a sorted array of size n, elements $z_1 < z_2 < \ldots < z_r$ can be searched for in $O(\log n + \log r)$ time using r processors on the EREW PRAM.*

Here is our parallel algorithm. In Step 1, for each column with index $c_i = i \times \log n$, where $i = 1, 2, \ldots, t(= n/\log n)$ use the multiple binary search algorithm to find row indices $p_i^{(j)}$, $j = 1, 2, \ldots, r$ such that

$$M[p_i^{(j)}, c_i] < z_j < M[p_{i+1}^{(j)}, c_i], \;\; j = 1, 2, \ldots, r.$$

At the end of the search retain the elements in the following submatrices.

$M[p_i^{(j)} + 1, p_{i-1}^{(j)} : c_{i-1}, c_i]$, $1 \le j \le r$, $1 \le i \le t$.

In Step 2, for each row with index $r_i = i \times \log n$, where $i = 1, 2, \ldots, , t(= n/\log n)$ use the multiple binary search algorithm to search for $z_1 < z_2 < \ldots < z_r$ in the retained elements in row r_i for column indices $q_i^{(j)}$, $j = 1, 2, \ldots, r$ such that

$$M[r_i, q_i^{(j)}] < z_j < M[r_i, q_i^{(j)} + 1], \;\; j = 1, 2, \ldots, r.$$

At the end of the search retain the elements in the following submatrices.

$M[r_{i-1}, r_i : q_i^{(j)} + 1, q_{i-1}^{(j)}]$, $1 \le j \le r$,$1 \le i \le t$.

Just before Step 3, each search has been narrowed down to $O(n/\log n)$ submatrices of size at most $\log n \times \log n$. However, each submatrix may have up to r searches remaining. The processors conducting the searches in the same submatrix have read conflicts which must be avoided. Let r' denote the number of searches (processors) sharing a submatrix. We have the following.

Case I. $r' = O(\log n)$. Use the simple pipelined algorithm of Section 3.1. The time required is $O(\log n + r')$, which is $O(\log n)$.

Case II. $r' = \Omega(\log^2 n)$. We can afford to sort the $O(\log^2 n)$ elements in the submatrix while retaining the original i, j indices for each element. Then use the multiple binary search algorithm to finish the search in $O(\log r + \log(\log^2 n))$ time.

Case III. $\log n < r' < \log^2 n$. Here we lack enough processors to sort the remaining elements in the submatrix while we have too many searches to use the simple pipelined algorithm. The key is applying the algorithm recursively to each such submatrix. In Step 1, let $t = \sqrt{\log n}$. Similarly take $\sqrt{\log n}$ rows and apply Step 2 appropriately. Now each remaining subsubmatrix has size at most $\sqrt{\log n} \times \sqrt{\log n}$. The number of searches left in each subsubmatrix is either $r' > \log n = (\sqrt{\log n})^2$ and Case II applies or the number of searches left $r' < \log n$ and Case I applies.

See [13] for run-time analysis and details.

Theorem 3.2 *Given an $n \times n$ sorted matrix M and r elements $z_1 < z_2 < \ldots < z_r$, the problem of multisearch and ranking for these r elements in the matrix M can be solved in $O(\log n + \log r)$ time using $O(rn)$ work on the EREW PRAM.*

4 Further Directions

I. Extend our algorithms to the coarse grained multicomputer model [16].

II. Consider Cartesian sets of the form $X + Y$ where the elements are $x_i + y_j$, with $x_i \in X, 1 \leq i \leq |X|$ and $y_j \in Y, 1 \leq j \leq |Y|$. If X and Y are sorted, then the resulting Cartesian set is a special case of a sorted matrix. We wish to extend our algorithms for multisearch in $X+Y$. For previous work for search, see Cosnard and Ferreira [3].

III. A important case is the multisearch for n elements in $X + Y$. The best-known sequential algorithm runs in $O(n^2)$-time [2]. No non-trivial lower bound is known [17]. Our ideas may lead to an $o(n^2)$ sequential algorithm. Improving the sequential algorithm is of great interest since it has ramifications for the knapsack problem [2, 18, 19].

References

[1] M. Cosnard, J. Duprat, and A. G. Ferreira. Known and new results in selection, sorting, and searching in sorted matrices and sorted $X+Y$. Technical Report LIP-IMAG 89-10, LIP - CNRS, Ecole Normale Supe′rieure de Lyon, France, 1989.

[2] M. Cosnard, J. Duprat, and A. G. Ferreira. The complexity of searching in $X + Y$ and other multisets. *Information Processing Letters*, 34:103–109, 1990.

[3] M. Cosnard and A. G. Ferreira. Parallel algorithms for searching in $X + Y$. In *Proceedings of the 18th International Conference on Parallel Processing*, pages III–16–III–19, 1989.

[4] G. N. Frederickson and D. B. Johnson. The complexity of selection and ranking in $X+Y$ and matrices with sorted columns. *Journal of Computer and System Sciences*, 24:197–208, 1982.

[5] G. N. Frederickson and D. B. Johnson. Finding kth paths and p-centers by generating and searching good data structures. *Journal of Algorithms*, 4(1):61–80, 1983.

[6] G. N. Frederickson and D. B. Johnson. Generalized selection and ranking: Sorted matrices. *SIAM Journal on Computing*, 13(1):14–30, 1984.

[7] D. Gries. *The Science of Programming.* Springer Verlag, 1981.

[8] A. Aggarwal, M. M. Klawe, S. Moran, P. Shor, and R. Wilber. Geometric applications of a matrix-searching algorithm. *Algorithmica*, 2:195–208, 1987.

[9] Joseph Ja′Ja′. *Introduction to Parallel Algorithms.* Addison Wesley, 1992.

[10] A. Aggarwal and J. Park. Notes on searching in multidimensional monotone arrays. In *Proceedings of 28th Conference on Foundations of Computer Science*, pages 497–512, 1988.

[11] Mikhail J. Atallah and S. Rao Kosaraju. An efficient parallel algorithm for the row minima of a totally monotone matrix. 13:394–413, 1992.

[12] Ramnath Sarnath and Xin He. Efficient parallel algorithms for selection and searching on sorted matrices. In *Proceedings of the International Parallel Processing Symposium*, pages 108–111, 1992.

[13] Amit Jain. Parallel search and multisearch in sorted matrices. Technical report, Boise State University, 1996. URL http://cs.idbsu.edu/∼amit/research/psmsm.html.

[14] R. J. Cole. Parallel merge sort. *SIAM Journal on Computing*, 17(4):770–785, August 1988.

[15] Danny Z. Chen. Efficient parallel binary search on sorted arrays. Technical Report 1009, Purdue University, Department of Computer Science, August 1990.

[16] F. Dehne, A. Fabri, and A. Rau-Chaplin. Scalable parallel geomtric algorithms for coarse grained multicomputers. In *ACM Symposium on Computational Geometry*, 1993.

[17] A. G. Ferreira. On space-efficient algorithms for certain np-complete problems. *Theoretical Computer Science*, 120:311–315, 1993.

[18] A. G. Ferreira. A parallel time/hardware tradeoff $T.H = O(2^{n/2})$ for the knapsack problem. *IEEE Transactions on Computers*, 40(2):221–225, 1991.

[19] E. Horowitz and S. Sahni. Computing partitions with applications to the knapsack problem. *Journal of the ACM*, 21(2):277–292, 1974.