High Speed Querying with the DAP 510

Peter J. Looges
Department of Computer Science
Old Dominion University
Norfolk, VA 23529

Abstract
Most database queries can be reduced to a few distinct classes of requests. These are (1) finding the \( k \)th smallest element in an unordered set \( S \), (2) finding the \( k \)th through the \( (k+i) \)th smallest elements in an unordered set \( S \), and (3) return the record with a specific key. We present a fast, efficient selection based system implemented on the DAP 510 for each of these query types.

1 Introduction
1.1 DAP Background
The DAP or the Distributed Array of Processors is a commercially available array processor, a product of Active Memory Technology. This massively parallel computer attaches to a host computer as a peripheral processor. It differs from the conventional serial processor in that it can perform the same operation on many items of data in parallel. Thus the DAP is a Single Instruction Stream, Multiple Data Stream (SIMD) architecture. For further details of SIMD and other architectural models, the reader is referred to [4].

The DAP 500 series machine is a 32x32 array of bit-organized processing elements, or PE's, each with its own memory of 64K bits. A PE has connections to each of its nearest neighbors and to a bus system which interconnects the PE's by row and column. It is these row and column busses which allow fast data fetching and broadcasting.

The DAP 510 is the specific model utilized, it has a cycle speed of 10 MHz, and can

- transfer between memory and processors at 1200MBytes/sec.
- accomplish Logic or Boolean operations at 10,000 million operations/sec.
- accomplish Character handling at 1000 million operations/sec.
- 8-bit integer add (2 address) at 400 million operations/sec[7].

The high level language available on DAP is known as APAL. This language is syntactically similar to FORTRAN, but has extensions which are designed to take full advantage of the DAP's matrix and vector processing facilities. An assembly language also known as APAL is also available on DAP. It is seldom necessary to use APAL for more than small subsections of DAP code. In fact, the DAP is a 'software based computer', it has little or no hardware committed to specialized operations such as floating point arithmetic and thus has much wider range of application than just numerical algorithms. Additional details of DAP applications are available in [8].

1.2 Previous Application of the DAP as a Filestore Search Engine
Page and Reddaway [6] examined the applicability of the ICL DAP-2 as a filestore search engine. Specifically, the DAP's memory organization and high speed input/output abilities are the keys highlighted. In addition, character keys and the location of specific keys in the database are the main points of [6]. We seek to extend the results of Page and Reddaway, additionally a faster DAP 510 was employed here.

2 Preliminaries
2.1 Single Element Selection
The process of finding the \( k \)th smallest element in an unordered set \( S \) has been studied through many theoretical models of parallel selection. R.Cole presented an optimal parallel algorithm requiring \( O(\log |S| \log^* |S|) \) time for selection on the theoretical EREW-PRAM model in [3]. Lin et. al. employ selection as an essential building block of their O(1) time sorting algorithm on the theoretical Mesh with Reconfigurable Buses [5].

In sequential processing, selection of the \( k \)th smallest element in the unordered set \( S \) is easily shown to require \( O(|S|) \) time and \( O(|S|) \) storage space [1]. Cole's selection algorithm for EREW-PRAM also requires \( O(|S|) \) storage space. The algorithm employs data movements that are consistent with the EREW-PRAM, but it is these data movements which cause the algorithm to not be directly applicable to most existing parallel architectures. With data movement being a key issue the optimal reconfigurable mesh algorithm of Lin et. al. is examined since reconfigurable mesh algorithms rely on efficient data movement. It quickly becomes apparent that this method is not applicable due to the number of required processing elements. The algorithm requires \( O(|S|^3) \) processing elements to solve the \( k \)th smallest query in O(1) time. If the original data set is \( O(1 \text{ million}) \) elements the storage requirement...
would be unrealistic for almost every implemented architecture in existence today. This would definitely be a requirement far exceeding that available on the DAP 510.

Thus our goal is to design an efficient selection algorithm that uses $O(|S|)$ storage and has limited data movement requirements for implementation on the DAP 510.

2.2 Range Selection

Range selection, the selection of the $k^{th}$ through the $(k + 1)^{th}$ elements from the set $S$, is a more complex version of single element selection. It entails the extraction of $i$ elements from set $S$. The difficulty of this is that these elements will, in most cases, not be consecutive in the unordered set $S$. Thus we not only need to be able to select a single item, but a group of $i$ randomly distributed elements. The same storage space constraints of single element selection apply.

2.3 Selection by Key

Efficient selection by key is most often done through the use of a semi-order (or full order) on the set $S$. A proper indexing scheme is known to be able to speed most accesses. This leads to fast retrieval as long as the key to the retrieval is the key by which the data is ordered. Multiple keys would either cause redundant portions of data or some retrievals being much slower than others. While this type of scheme is potentially very efficient for retrieval if can place a high cost on storage (both time and space) in a particularly dynamic environment. Thus if an efficient mode of retrieval can be obtained for totally unordered sets of keys without increasing storage requirements the overall efficiency of the data system will be considerably improved.

The remainder of this paper is organized as follows, section 3 presents the DAP algorithms. Section 4 discusses the results of the implementation of this algorithm.

3 DAP Algorithms

As we have noted, the goal is to produce algorithms that require not only a linear quantity of storage, but also one which solves the $k^{th}$ smallest element in $S$ query efficiently. To do this we exploit the associative nature of the DAP's memory and the efficient use of logical masks to control computation of basic operations. Recent improvements in the DAP's procedures for mapping vectors larger than the processor mesh on to this fixed size mesh definitely aid the efficiency of these algorithms.

3.1 Single Element Selection

We employ the DAP routines MINV, MAXV, SUM which produce the obvious results in that MINV returns the Value of the minimum entry in the supplied vector. A second argument for the function allows a logical mask to be supplied to the routine, in this case if entry $i$ of the mask is TRUE, then entry $i$ of the argument vector is examined, if FALSE the entry in the argument at the $i^{th}$ position is ignored. The SUM routine simply sums the returns the sum of all entries in its argument. In the case of a logical vector, TRUE is treated as 1 and FALSE is treated as 0. Thus we implement the selection as a binary search on the unordered vector, through the use of logical masks. For the average case, each iteration reduces the search area size by roughly $\frac{1}{2}$. Later we show that the average case may actually be considered as worst case for the DAP implementation. We consider all values which are not masked off to be in the active set. The function select follows:

\[
\text{Integer Function Select}(V,K); \\
\{\text{Input: Vector V, Integer K that causes the selection of the } k^{th} \text{ element of V} \} \\
\text{Output: The } k^{th} \text{ smallest element of V} \}
\]

begin \\
Mask ← TRUE; \{Mask is logical vector of the same length as V\} \\
min ← \text{MINV}(V); \\
max ← \text{MAXV}(V); \\
begin loop forever; \\
\text{midvalue} ← \frac{\text{min} + \text{max}}{2}; \\
\text{Totalless} ← \text{SUM}((V < \text{midvalue}) \land \text{AND}\text{Mask}); \\
\text{If} \text{Totalless} < K \text{ then begin} \\
K ← K - \text{Totalless}; \\
\text{min} ← \text{midvalue}; \\
\text{Mask}(V < \text{midvalue}) ← \text{FALSE} \\
\text{end; Else begin} \\
\text{max} ← \text{midvalue}; \\
\text{Mask}(V > \text{midvalue}) ← \text{FALSE} \\
\text{end;}
\end{loop} \\
end; \{\text{function select}\}

Theorem 1. Function Select returns the $k^{th}$ smallest object from the vector $V$. □

The only problem with this algorithm is that given a particularly pathological data set, the computation time could be $O(|V|^2)$. This could be a serious problem given the excessively large data sets we intend to handle with these algorithms. In the following theorem we show that this case does not apply and therefore need not be considered.

Theorem 2. With function Select we need not consider the pathological data case on the DAP.

Proof. First, we must establish what this "pathological" data set is: A data set which would cause only a very few elements to be eliminated at each iteration. This means that if the data set was sorted each successive element would be at least as large as the sum of all preceding elements. This would be the case if all the elements of the data set were all powers of 2. On the DAP the maximum number of bits that may represent a number (real or integer) is 64. Thus for integers this is 64 possible elements for this data set, for real values this increases to about 150 due to scientific notation. The largest value representable is $O(10^{78})$. 

343
Thus both pathological sets, integer and real, are very small indeed. Given that we can sort 1,000 integers in less than .01 seconds [2] these small data sets need not be considered. □

This selection may be considered to be an implementation of binary search on an unordered set $S$, in that, at each iteration the goal is to reduce the size of the search area by 50%. This may not always occur, but for there to be a truly noticable reduction in performance we would need to return to the pathological data case.

3.2 Range Selection

As noted earlier, range selection is a more complex version of single element selection. One method of doing this would be to run single element selection $i$ times. This method would obtain the desired result, but most probably not in the most efficient manner. It is much more efficient to extract the $k^{th}$ and $(k+i)^{th}$ keys first, then simply rely on the DAP's associative memory and extract all keys which lie between the two extreme keys. Procedure Range.Select provides the details:

```
Procedure Range.Select(V,K,I);
{Input: Vector V, Integers K and I that causes the selection of all elements from the $k^{th}$ through the $(k+i)^{th}$ elements of V
Output: The $k^{th}$ through $(k+i)^{th}$ smallest elements of V
}
begin
if($I < 0$ or $(K+1 > LENGTH(V)))$ return('Invalid Range');
RESULT.KEYS ← 0
KEY1 ← SELECT(V,K); KEY2 ← SELECT(V,K+I);
RESULT.KEYS((KEY ≥ KEY1) and (KEY ≤ KEY2)) ← KEY;
COMPACT(RESULT.KEYS);
return(RESULT.KEYS)
end; {procedure Range.Select}
```

Theorem 3. Procedure Range.Select returns the $k^{th}$ through $(k+i)^{th}$ keys (elements) of the vector V. □

3.3 Selection by Key

Range Selection shows how direct selection is once the key is known. The DAP's associative memory and ability to efficiently map vectors which exceed the array size by factors of 1000 or more, provided sufficient memory is available. Thus the identification of a specific record which holds some key is a matter of a search on that key range for that key. Due to the basic nature of the process the procedure will not be given, but results for this type of selection are shown later.

4 Selection Results

The implementation of these routines has provided some interesting results, as shown in figure 1. It is especially encouraging to note that for sets of 1,000,000 elements the response is so fast.

Figure 1: Comparison of Selection Results

The data demonstrates that the procedure is consistent across a wide range of input data sizes. Computation time increasing with the input size as would be expected. Figure 1 shows the stability of the procedure, in that the computation time is reasonably consistent with in the same size data set, no matter the element selection

Selection by a known key would be expected to be the most efficient given the associative nature of the DAP's memory. The key selection line in figure 1 supports this expectation.

5 Conclusion and Open Questions

We have developed a very fast efficient selection algorithm for the DAP 510. While it does not support all possible cases of data, it's average case performance makes it a viable selection technique for the DAP. Since the data set required to degrade performance is of little actual concern this method is sufficiently powerful to be a viable tool for query answering when the DAP is employed as a filestore search engine.

A question that remains open is on of multi-selection. Multi-selection is the processing of a sequence of queries, where the only known relation between the queries is that they are ordered in some manner. A direct solution to this is just processing each query independently, but this does not seem to be the most efficient solution, just as it was not the most efficient for range selection. We are continuing to explore this type of multiple querying for implementation on the DAP 510.

References


