Parallel Algorithms for Indexing and Retrieval in Audio Databases

S.R. Subramanya Abdou Youssef
Department of Electrical Engineering and Computer Science,
The George Washington University,
Washington, DC 20052.

Abstract

Recent explosion in the use of non-text, *multimedia data* such as audio, video, images, and graphics necessitates the development and use of multimedia databases. Efficient schemes for indexing and searching in multimedia databases are essential for fast and sophisticated data retrieval. The highly complex nature of audio/visual query processing, and the vast amounts of data in multimedia databases, require the use of parallel algorithms for indexing and searching to achieve the desired efficiency in data retrievals. Indeed, high-performance parallel processing systems are beginning to be used as servers for multimedia databases, alongside workstations and PCs with multiple processors, as clients. While some work has been done on image and video databases, little attention has been paid to audio databases. In this paper we design and analyze two parallel algorithms for searching for audio data in audio/multimedia databases, using a transform-based indexing scheme.

1 Introduction

The digital storage and processing of non-textual data, such as *video*, *audio*, and *images*, commonly referred to as *multimedia data* has grown tremendously in recent times and is expected to grow at the same or faster rate in the coming years [6, 7, 11, 12, 13]. The huge sizes and *complex* nature of multimedia data have rendered *keyword-based* queries and *exact* searches used in traditional databases ineffective, and have called for *query-by-content*(QBC), *query-by-example*(QBE) and *similarity* searches [1].

A query-by-example in an audio database takes a sound sample as the query and finds all files containing an exact match or, more often than not, a close resemblance to the query sample. Applications include (1) music "sound-likes" for music listeners, entertainment professionals, and music experts; (2) music sound-likes for detection of plagiarism and music copyright violations; (3) identification of explosion types by matching the heard explosion sound against a database of known recorded explosion sounds; (4) security applications such as personal identification by matching a person's voice against a database of voices of authorized personnel; and (5) law enforcement such as criminal identification by matching a suspect's phone-recorded voice against a database of recorded voices.

The aforementioned applications of queries by example in audio databases make it amply clear that we need novel and efficient ways for indexing and similarity searching in multimedia databases. Although there have been several efforts toward the development of databases of images [2, 3, 4, 5, 6, 7, 9, 10] and video [11, 12, 13], not much attention has been given to audio data [16, 14, 15]. Audio data could serve as an independent data type or as part of multimedia data.

A transform-based indexing scheme and two sequential search algorithms for audio databases has been proposed in another paper [15]. In this paper, we design and analyze two *parallel algorithms* on a PRAM model for similarity searching, based on the transform-based indexing scheme. The analysis will show that the parallel algorithms are quite fast and exhibit linear speedup.

2 Transform-based indexing

A transform-based indexing for audio data for use in multimedia databases has been described in [15] and only an outline is given in this section. The transform-based indexing scheme offers several advantages such as (1) insensitivity to both noise and varying sampling rates, and (2) use of only a small number of transform coefficients to characterize the data.

Our transform-based indexing approach can be outlined as follows. Each audio file or stream is divided into small blocks of contiguous samples and a transform like the Discrete Fourier Transform (DFT) or the Discrete Cosine Transform (DCT) is applied to each

block. This yields a set of coefficients in the frequency domain. With a suitable transform, only a few significant coefficients are adequate to reconstruct a good approximation of the original signal.

An index entry is created by selecting an appropriate subset of the transform coefficients and retaining a pointer to the original audio block. Thus, the index occupies much less space than the data and allows for faster searching. When a query by example is to be processed, the query is similarly divided into blocks, each is transformed, and a subset of transform coefficients is selected. This forms the pattern. Then, the index data is searched for an occurrence of this pattern within some measure of similarity. In this case, two strings are considered matched if they are within a small enough 'distance' of each other, where the distance is the root-mean-square-difference of the significant transform coefficients.

It is to be noted that in our scheme described above, the data and the query are divided into blocks, before applying transforms because of the several advantages offered by blocking such as the following: (1) When transforms are applied to the whole signal, the transform coefficients capture global averages but not the the finer details. (2) Blocks of appropriate sizes would contain samples which are highly intercorrelated, so that when transforms are applied, there is more energy compaction and thus fewer transform coefficients would adequately describe the data. (3) The transforms on the individual blocks could be carried out in parallel.

3 Sequential Search Algorithms and their Analyses

Two sequential search algorithms developed for the transform-based indexing – *index search* schemes are described and analyzed in [15]. These are briefly given in this section to aid in the understanding of the parallel versions of these algorithms. The performance of the sequential search algorithms are compared with *raw-data search*, where samples of the given query are compared against the data samples in the audio files.

The raw-data search is rather naive. It determines if there is a close match between the query samples and any portion of the data. Specifically, let the query be $Q = q_1 q_2 \cdots q_m$ and the data be $A = a_1 a_2 \cdots a_n$. A close match exists if there is a sequence $a_{k+1} a_{k+2} \cdots a_{k+m}$ in A such that $\sqrt{\sum_{i=1}^{m} (a_{k+i} - q_i)^2} < \mathcal{E}$, where $\mathcal{E}$ is some given threshold.

In the index searches, the corresponding transform coefficients of the query are compared with the coefficients of the data blocks which have been selected and retained to serve as indices, and the Euclidean distance between them is determined. If the distance is below an experimentally determined threshold, it is accepted as a match. The first of the two index search schemes is called *SimpleSearch*. It assumes that the query block boundaries are aligned with those of the data block boundaries. The other algorithm called *RobustSearch* removes this restriction so that the query blocks could be at any part of the data.

In the following algorithms and their analysis, the following notations are used:

L:	The length of a block (Number of samples in a block).
N, M:	Number of blocks of the data and the query, respectively.
k:	Number of significant transform coefficients per block retained as index (for that block).
P:	The number of Processing Elements (PEs).
QBC:	Query Block Coefficients (The coefficients obtained after applying transform on blocks of the query.
DBC:	Data Block Coefficients (The k largest coefficients obtained after applying transform on blocks of the original data.
$DBCL$:	Locations of the k largest Data Block Coefficients.
RBC:	Reconstructed Block Coefficients.
$RBCL$:	Reconstructed Block Coefficient Locations.

Note that each block of QBC contains L elements and each block of DBC and $DBCL$ contains k elements. All logarithms used in the analysis are to base 2. In the algorithm descriptions, the phrase 'matching query blocks with data blocks' refers to finding the Euclidean distance between the selected coefficients of data blocks (indices) and the corresponding coefficients of query blocks.

3.1 Simple search algorithm

As mentioned earlier, it assumes that the query block boundaries are aligned with those of the data block boundaries.

The algorithm BLOCKDISTANCE determines the Euclidean distance between the corresponding coefficients of a block of query and the coefficients of a block of data, and is used in all the following search algorithms.

Algorithm 3.1 BLOCKDISTANCE (QBC, j, DBC, l)
{Block j of QBC is matched with block l of DBC.}

```
1.  begin
2.    dist ← 0;
3.    for i = 1 to k do
4.      loc ← DBCL[(l − 1)L + i];
5.      c1 ← DBC[(l − 1)L + i];
6.      c2 ← QBC[(j − 1)L + loc];
7.      dist ← dist + (c1 − c2)^2;
8.    endfor
9.    return(sqrt(dist));
10. end
```

Algorithm 3.2 SIMPLESEARCH $(QBC, DBC, DBCL)$

```
1.  begin
2.    for i = 1 to N − M do
3.      dist ← 0;
4.      for j = 1 to M do
5.        dist ← dist+BLOCKDISTANCE
              (QBC, j, DBC, i + j − 1);
6.      endfor
7.      if (dist < Threshold)
8.        Report 'Match Found' and
          break from loop.
9.      endif
10.   endfor
11.   Report 'Match Not Found'.
12. end
```

3.2 Robust search algorithm

As mentioned earlier, the query blocks could be at any part of the data, not necessarily aligned with data block boundaries, and the the robust search algorithm is capable of handling this. A brief description of the algorithm is now given followed by the pseudocode. Recall that L is the block length (number of samples in a block), and M is the number of blocks in the query.

Before matching all the query blocks with the data blocks, only the first block of the query is matched against successive data blocks to determine the position in the data block where the query is possibly aligned. This is done by taking the inverse transform of two successive data blocks (length $2L$), and then by considering a window of L samples and sliding it so that the beginning of the window moves from the first to the last sample of a block. At each step, a transform of the window is taken and is matched against the first block of the query. The place where the distance between the data and query block is minimum is recorded, say r. If the minimum distance is less than a suitable, empirically determined threshold, then the position where it occurs is tentetaively accepted as a possible position where the query matches the data. Subsequently, the whole query is matched against the data blocks, aligned at that position. This is done by taking the inverse transform of $M+1$ data blocks and then by considering r as the beginning of a block and taking the transform of M blocks, and then matching these blocks with the query blocks. In case of a match, it is reported; otherwise, the process repeats with the next two consecutive blocks.

Algorithm 3.3 ROBUSTSEARCH ($QBC[1 : M, 1 : L]$, $DBC[1 : N, 1 : k]$, $DBCL[1 : N, 1 : k]$)

```
1.  begin
2.    n ← 1;
3.    while n ≤ N − M do
4.      Apply the inverse transform on DBC[n]
        and DBC[n + 1] to get R_1, an
        approximation of the original signal
        of length 2 blocks (2L samples).
5.      for k = 1 to L do
6.        W = Transform(R[k : k + L − 1]).
7.        Compute the Euclidean distance
          between corresponding coefficients
          of W and QBC[1]. Save the minimum
          distance so far in min_dist and the
          position where it occurs in r.
8.      endfor
9.      if (min_dist < Threshold2)
          (Accept tentatively).
10.       Apply inverse transform on blocks
          DBC[n : n + M + 1] to get R_2, an
          approximation of the original signal,
          of length M + 1 blocks. (Next align
          R_2 with the query where a tentative
          match occurred).
11.       Consider R_2[r : r + M · L] and divide
          this into M blocks. Apply transform to
          each of the blocks to get RBC[1 : M]
          and RBCL[1 : M].
12.       dist ← 0;
13.       for i = 1 to M do
14.         dist ← dist+BLOCKDISTANCE
                      (QBC, i, DBC, n + i − 1);
15.       endfor
16.       if (dist < Threshold1)
17.         Report 'Match Found' and
            break from loop.
18.       else n ← n + M;
19.       endif
20.     else n ← n + 1;
21.     endif
22.   endwhile
23. end
```

3.3 Analysis of sequential search algorithms

The detailed analysis of the sequential simple search and robust search algorithms are given in [15] and are summarized below. The time complexity of the simple search algorithm is $\approx \mathcal{O}(ML \log L + NMk)$, and its speedup over raw-data search, S_1 is $\approx L^2/k$.

The time complexity of the robust search algorithm is $\approx \mathcal{O}((N - M)(L^2 \log L))$, and its speedup over raw-data search, S_2 is $\approx \frac{NM}{(N-M) \log L}$

4 Parallel Search Algorithms and their Analyses

In this section, we design two parallel query matching schemes, based on the sequential SIMPLE SEARCH algorithm. We then analyze the time complexities of the parallel algorithms and determine their speedups over the sequential version.

We assume a PRAM-CREW (Parallel Random Access Memory- Concurrent Read Exclusive Write) model. In this model (described in [17]), a set of P processing elements (PEs) are connected to a shared memory through an interconnection network. Any memory location can be accessed for write by only one PE at any time. However, several PEs could read one memory location concurrently, and can write to different locations concurrently. Note that in PRAM-EREW (Exclusive Read Exclusive Write), the only difference is that a memory location can be read by only one PE at a time. If several PE's need to read the same location, as many copies of that location will have to be made in logarithmic time. To make our algorithms implementable on both models, we will indicate which variable(s) will be copied and into how many copies. In the time analysis, this copying (or broadcasting) is assumed to take constant time in CREW but logarithmic time in EREW.

4.1 Parallel Simple Search Algorithm

The SIMPLESEARCH algorithm can be parallelized in several ways, each leading to different speedup. For the sake of brevity, we present the parallel scheme that exhibits linear speedup.

In that scheme, each PE has its own copy of the query coefficients (QBC) (or all can read concurrently from a single copy). All the PEs act in parallel, with each PE trying to match in sequence, the blocks of the query coefficients with the data block coefficients, shifted appropriately, as shown in Figure 1. Note also that the PEs read blocks of DBC concurrently, which is allowed in the CREW model.

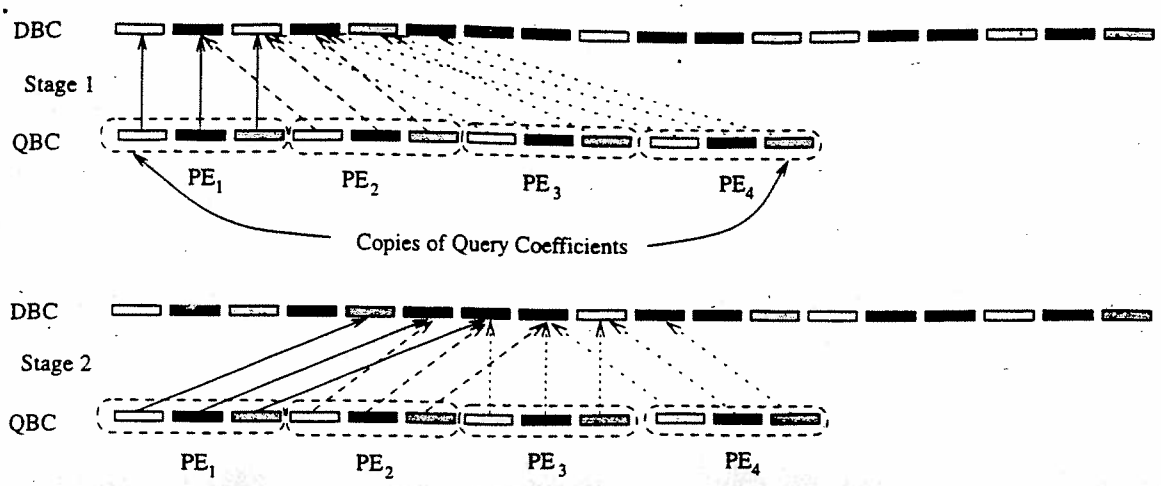

Figure 1: Parallel simple search: Each PE works on the whole query. The figure shows two successive stages in the matching process.

Algorithm 4.1 PARALLELQBESEARCH **(in: Q, DBC, DBCL)**

1. **begin**
2. PARALLELTRANSFORMQUERY*(Q, QBC).*
3. *Make P copies of QBC so that each PE_i has its local copy of QBC.*
4. PARALLELSIMPLESEARCH1 *(QBC, DBC, DBCL).*
   { *Or* PARALLELSIMPLESEARCH2 *(QBC, DBC, DBCL).*}
   { *Or* PARALLELROBUSTSEARCH *(QBC, DBC, DBCL).*}
5. **end**

Algorithm 4.2 PARALLELTRANSFORMQUERY **(Q, QBC)**

1. **begin**
2. **for** $i = 1$ **to** P **pardo**
3. *Assign blocks $(i-1)\lceil\frac{M}{P}\rceil + 1$ to $i\lceil\frac{M}{P}\rceil$ to PE_i to perform DCT on each block;*
4. **endfor**
5. **end**

Algorithm 4.3 PARALLELSIMPLESEARCH1 *(QBC, DBC, DBCL)*

1. **begin**
2. **for** $n = 1$ **to** $\lceil\frac{N-P}{P}\rceil$ **do**
3. **for** $i = 1$ **to** P **pardo**
   PE_i (does):
4. $dist[i] \leftarrow 0$;
5. **for** $j = 1$ **to** M **do**
6. $l \leftarrow (n-1)P + i + j - 1$;
7. $dist[i] \leftarrow dist[i]+$ BLOCKDISTANCE(QBC, j, DBC, l);
8. **endfor**
9. **if** $dist(i) < Threshold$)
10. *Report 'Match Found' and* `exit`.
11. **endif**
12. **endfor**
13. **endfor**
14. *Report 'Match Not Found'.*
15. **end**

4.1.1 Analysis of the Parallel Simple Search Algorithm

We analyze the time complexity on the CREW model and determine the speedup. DCT application on each block of the query of size L takes $\mathcal{O}(L\ log\ L)$ time. Since each PE transforms $\lceil\frac{M}{P}\rceil$ blocks, the time for PARALLELTRANSFORMQUERY is $\lceil\frac{M}{P}\rceil L\ log\ L$. Step 7 of PARALLELSIMPLESEARCH1 takes $\mathcal{O}(k)$ time, and the for-loop of steps 5–8 thus takes $\mathcal{O}(Mk)$ time. Therefore, the whole PARALLELSIMPLESEARCH1 takes $\mathcal{O}(\lceil\frac{N-P}{P}\rceil Mk)$. Consequently, the total time of the PARALLELQBESEARCH algo-

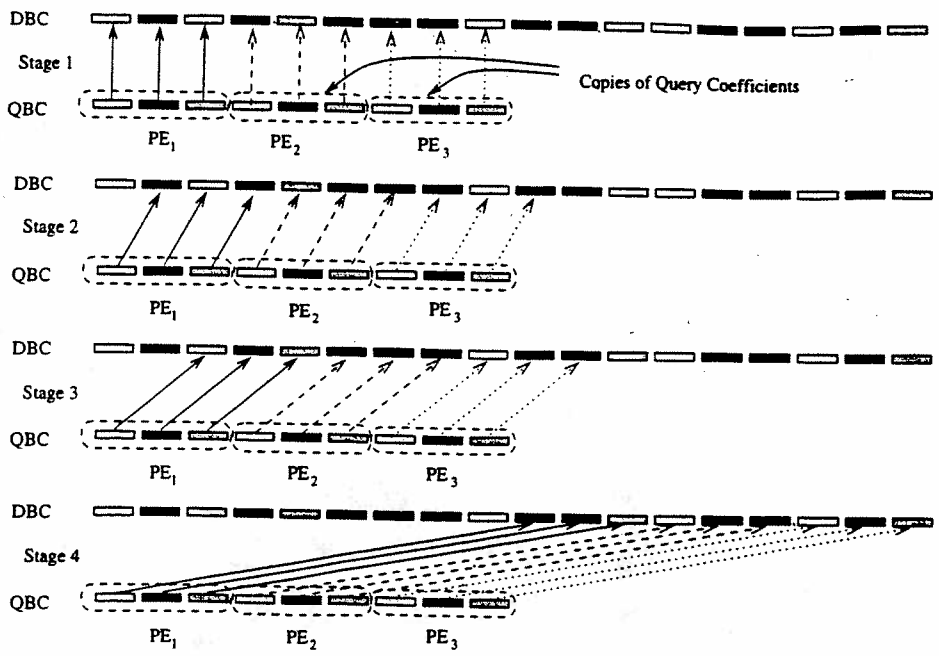

Figure 2: Parallel simple search on EREW model. Each PE works on the whole query. Figure shows four successive stages in the matching process.

rithm is:

$$\mathcal{O}(\lceil\frac{M}{P}\rceil L \log L + \lceil\frac{N-P}{P}\rceil Mk)$$

The speedup of the PARALLELQBESEARCH algorithm over the sequential SIMPLESEARCH algorithm is:

$$\frac{\mathcal{O}(ML \log L + NMk - M^2k)}{\mathcal{O}(\lceil\frac{M}{P}\rceil L \log L + \lceil\frac{N}{P}\rceil Mk)}$$

which is clearly a linear speedup:

$$Speedup = \mathcal{O}(P).$$

4.2 Parallel Simple Search Algorithm on EREW model

We describe an alternate parallel search algorithm on an EREW model. In this algorithm, P copies of the query coefficients (QBC) are made and each copy is treated as local to each of the PEs. All the PEs act in parallel, with each PE trying to match the blocks of the whole query with the data block coefficients, shifted appropriately, as shown in Figure 2.

Algorithm 4.4 PARALLELSIMPLESEARCH2 (QBC, DBC, $DBCL$)

```
1.  begin
2.    for n = 1 to ⌈N/P⌉ do
3.      for i = 1 to P pardo
4.        PE_i (does):
5.        for j = 1 to M do
6.          l ← (n − 1)P + i + j;
7.          dist[i] ← dist[i]+BLOCKDISTANCE
                  (QBC, j, DBC, l).
8.        endfor
9.      endfor
        {Check if there is any match }
10.   for i = 1 to P pardo
11.     if (dist[i] < Threshold)
12.       found[i] ← TRUE;
13.     endif
14.   endfor
15.   match ← AND_{1≤i≤P}(found[i]),
      using reduction.
16.   if (match = TRUE)
17.     Report 'Match Found' and
        break from loop.
18.   endif
19. endfor
20. Report 'Match Not Found'.
21. end
```

4.3 Analysis of Parallel Simple Search Algorithm on EREW model

Making P copies of QBC takes $\mathcal{O}(ML \log P)$ time, using broadcast. (Note that QBC is of size ML elements). DCT application on each block of the query of size L takes $\mathcal{O}(L \log L)$ time. Since this is done in parallel $\lceil \frac{M}{P} \rceil$ times, the time for ParallelTransformQuery is $\lceil \frac{M}{P} \rceil L \log L$.

BlockDistance takes $\mathcal{O}(k)$ time. Since this is called within the `for` loop M times, the time for this `for` loop of steps 5–9 is $\mathcal{O}(Mk)$. The parallel loop of steps 3–10 also takes $\mathcal{O}(Mk)$ time. Steps 11–15 take $\mathcal{O}(1)$ time. Step 16 takes $\mathcal{O}(P)$ time, the time for the parallel reduction operation. So, within the outer `for` loop from 2–20, each iteration takes $\mathcal{O}(Mk + log\ P)$ and the loop is iterated $\lceil \frac{N}{P} \rceil$ times. So, the time for the search is: $\mathcal{O}(\lceil \frac{N}{P} \rceil (Mk + log\ P))$.

Taking into account the time for the initial transform of the query and the copying of query coefficients, the total time of the ParallelQBESearch algorithm using ParallelSimpleSearch2 is:

$$\mathcal{O}(\lceil \frac{M}{P} \rceil L \log L) + \mathcal{O}(ML \log P) + \mathcal{O}(\lceil \frac{N}{P} \rceil (Mk + \log P)) .$$

The speedup of the ParallelQBESearch algorithm using ParallelSimpleSearch2, over the sequential SimpleSearch algorithm is:

$$\frac{\mathcal{O}(ML \log L + NMk - M^2k)}{\mathcal{O}(\lceil \frac{M}{P} \rceil L \log L) + \mathcal{O}(ML \log P) + \mathcal{O}(\lceil \frac{N}{P} \rceil (Mk + \log P))}$$

4.4 Parallel Robust Search Algorithm

As mentioned in Section 3.2, the simple search algorithms require that the query blocks be aligned with the data blocks for correct retrievals. This limitation is overcome by the robust algorithm. Refer to the description of the sequential robust search algorithm given in Section 3.2. In the parallel version of the robust search algorithm, we assume that $P = MAX(L, M)$. Recall that P is the number of PEs, L is the block length, and M is the number of blocks in the query. Generally, $L > M$.

Algorithm 4.5 ParallelRobustSearch (QBC, DBC, $DBCL$)

1. **begin**
2. $n \leftarrow 1$;
3. **while** $n \leq N - M$ **do**
4. *Apply the inverse transform on blocks n and $n+1$ of DBC* **in parallel** *to get R_1 an approximation of the original signal of length 2 blocks ($2L$ samples).*
5. *Make L copies R_{1i} $(1 \leq i \leq L)$ of R_1 and the first blk of QBC, using broadcast.*
6. **for** $i = 1$ **to** L **pardo**
7. $W_i \leftarrow Transform(R_{1i}[i : i + L - 1])$.
8. $d1[i] \leftarrow$ BlockDistance $(QBC, 1, W_i, 1)$; {*Compute the Euclidean distance between corresponding coefficients of W_i and local copy of $QBC[1]$*}
9. **endfor**
10. $min\_dist \leftarrow MIN_{1 \leq i \leq L}(d1[i])$; {*Using Parallel Reduction*}. *Record the value of i where the minimum occurs, in r.*
11. **if** *$(min\_dist < Threshold2)$ (Accept tentatively).*
12. **for** $i = 1$ **to** $M + 1$ **pardo**
    PE_i *(does):*
13. *Apply inverse transform on block $n + i - 1$ of DBC to get $R2_i$, an approx. of the original signal block.*
14. **endfor**
15. *Align $R_2 = CONCAT_{1 \leq i \leq M+1}(R_{2i})$ with the query where a tentative match occurred to get R'_2. Divide the aligned R'_2 into M blocks, R'_{2i}, $1 \leq i \leq M$.*
16. **for** $i = 1$ **to** M **pardo**
    PE_i *(does):*
17. *Apply transform to block R'_{2i} to get RBC_i and $RBCL_i$.*
18. $d2[i] \leftarrow$ BlockDistance (QBC, i, RBC, i);
19. **endfor**
20. $dist \leftarrow ADD_{1 \leq i \leq M}(d2[i])$; {*Using Parallel Reduction.*}
21. **if** *$(dist < Threshold1)$*
22. *Report 'Match Found' and* `break` *from loop.*
23. **else** $n \leftarrow n + M$;
24. **endif**
25. **else** $n \leftarrow n + 1$;
26. **endif**
27. **endwhile**
28. **end**

4.4.1 Analysis of the Parallel Robust Search Algorithm

We analyze the time complexity on the CREW model and determine the speedup. The following table gives the complexities of different logical units of the algorithm.

Steps	Time	Steps	Time
4–6	$\mathcal{O}(L \log L)$	8–11	$\mathcal{O}(L \log L + k)$
12	$\mathcal{O}(\log L)$	14–16	$\mathcal{O}(L \log L)$
17	$\mathcal{O}(1)$	18–21	$\mathcal{O}(L \log L + k)$
22	$\mathcal{O}(\log M)$		

As in the sequential case, there are three cases and the worst case occurs when there is no tentative match. The time time in this case would be: $(N - M)\mathcal{O}(L \log L + k)$. Taking into account the $\mathcal{O}(L \log L)$ time for transforming the query in parallel, the total time complexity for the algorithm is (still): $(N - M)\mathcal{O}(L \log L + k)$. So, the speedup S_{rbst} over the sequential version is:

$$\approx \frac{\mathcal{O}(ML \log L) + (N - M) \cdot \mathcal{O}(L^2 \log L + Lk)}{(N - M)\mathcal{O}(L \log L + k)}$$

Generally, $k \ll L \log L$. So, the speedup is:

$$\approx \mathcal{O}(\frac{M}{N - M} + L + \frac{k}{\log L}) \approx \mathcal{O}(L).$$

Note that we have assumed $P = MAX(L, M)$. Since (generally), $L > M$, the speedup is $\mathcal{O}(P)$, which is linear.

5 Conclusions and Future directions

A transform-based indexing scheme and sequential algorithms for searching for audio data in audio/multimedia databases were proposed in [15] and shown to be quite effective. This paper proposed and analyzed parallel versions of the search algorithms and showed that they have linear speedup. This effectiveness, and the efficient parallel algorithms for searching presented in this paper, make our indexing and searching schemes quite attractive for audio databases.

Our future work include the implementation and testing of the algorithms on actual parallel client-server systems, and extending our techniques to other types of multimedia data such as images and videos.

References

[1] Narasimhalu, A.D. ed. Special issue on content-based retrieval. *ACM Multimedia systems*, Vol. 3, No. 1, Feb 1995.

[2] Alexandrov, A.D. *et.al.* Adaptive filtering and indexing for image databases. *SPIE*, Vol. 2420, pp. 12-22.

[3] Chang, S-K. 'Image Information Systems', *Proc. IEEE*, Vol. 73, No. 4, April 1995, pp 754-764.

[4] D'Allegrand, M. Handbook of image storage and retrieval systems. *Van Nostrand Reinhold*, New York, 1992.

[5] Gong, Y., *et.al.* 'An Image Database System with Content Capturing and Fast Image Indexing Abilities', *Proc. Int'l Conf. on Multimedia Computing and Systems*, 1994, pp121-130.

[6] Grosky, W. and Mehrotra, R. eds. Special issue on image database management. *IEEE Computer*, Vol. 22, No. 12, Dec 1989.

[7] Gudivada, V. and Raghavan, V. Special issue on content-based image retrieval systems. *IEEE Computer*, Sept. 1995, Vol. 28, No. 9.

[8] Idris, F. and Panchanathan, S. 'Storage and Retrieval of Compressed Images', *IEEE Trans. on Consumer Electronics*, Aug. 1995, pp937-941.

[9] Jain, R. *et. al.* Similarity measures for image databases. *SPIE*, Vol. 2420, pp. 58-61.

[10] Petrakis, E.G.M. and Faloutsos, C. 'Similarity Searching in Large Image Databases', *UMD Technical Report.*

[11] La Casia, M. and Ardizzone, E. 'Jacob: Just A Content-based Query System for Video Databases', *IEEE Multimedia Conf.*, 1996 pp 1216-1219.

[12] Hampapur, A., *et. al.* 'Digital Video Segmentation', *ACM Multimedia 94 Conf. Proc.*, pp 357-364.

[13] Smoliar, S.W. and Zhang, H.J. 'Content Based Video Indexing and Retrieval', *IEEE Multimedia*, Summer 1994, pp62-72.

[14] Ghias, A. *et. al.* Query by humming. *Proc. ACM Multimedia Conf.*, 1995.

[15] Subramanya, S.R. *et. al.* 'Transform-Based Indexing of Audio Data for Multimedia Databases', *IEEE Int'l Conference on Multimedia Systems*, Ottawa, June 1997.

[16] Wold, E. et al. Content-based classification, search and retrieval of audio data. *IEEE Multimedia Magazine*, 1996.

[17] Quinn, M.J. *Parallel Computing - Theory and Practice*, McGraw-Hill, 1994.