Parallelising Speech Recognition Algorithms

Abstract

A variety of current speech algorithms are discussed and shown to exhibit much potential for parallel execution. The different processing requirements of recognition and training are shown to map efficiently onto SIMD and MIMD architectures respectively. The PADMAVATI architecture under development in IPSD supports both forms of parallelism. An associative dialect of "C" is presented and used to implement a One Pass connected speech recognition algorithm using the PADMAVATI content addressable memory (CAM). Performance estimates show that this algorithm will run faster than real time on the PADMAVATI architecture. A scheme is presented to perform training and recognition runs in parallel on PADMAVATI using the farm paradigm. The scheme requires only a minimal software investment yet should yield a near-linear speed-up with increasing numbers of processor nodes.

Circulation

Abstract only: Dr W S Bardo ) GEC-Marconi Ltd, Stanmore

Mr D G Barlow )
Dr B J Isherwood )
Mr D L Lewis ) Hirst Research Centre
Dr E Read )
Dr D G Scotter )
Dr I R Young )
Dr M A El Fatatry )

MRC Library ) Marconi Research Centre, Baddow

ERC Library ) Engineering Research Centre, Stafford
Parallelising Speech Recognition Algorithms

Present: B.L.R. Cameron
D.J.P. Pearce (1st meeting)
L.C. Wood
D.B. Howe
J.D. Robson
K.L. Griffiths (1st meeting)
K. Asanovic

Introduction

These meetings were held to consider the potential for using parallel architectures to accelerate speech algorithm execution. This report is structured in three sections. The first section reviews current speech recognition algorithms as presented by LCW. The second section outlines currently available parallel systems, and describes the PADMAVATI architecture under construction within IPSD. The final section discusses how speech algorithms could be executed on parallel systems, with particular reference to the PADMAVATI architecture.

1. Speech Algorithms

All those considered had two phases, namely training and recognition. During training a set of reference models is constructed from a set of training utterances. We may choose to model whole words or smaller sub-word units. In the recognition phase an unknown input utterance is compared against the reference models. Where the input is constrained to be a single word, and where our reference models are also single words, we can select the reference which best matched the entire input utterance as our recognised word. In other cases we need to select the best sequence of reference models as our recognised word or sentence.

The following sections discuss the various models used, their training algorithms, and the corresponding recognition algorithms.

Reference Templates

One approach to modelling a reference speech pattern is to use a template consisting of a time sequence of frames. Each frame is a vector of parameters which characterise a short section of speech (~10-20 ms duration). The reference templates are trained by some form of averaging over a set of training utterances.

January 26, 1989
Template Recognition With Dynamic Time Warping (DTW)

DTW is a method used to recognise reference templates. Here we consider the simplest case; recognising an isolated word with whole word reference templates. First we introduce some notation (taken from [1]).

We label frames in the input sequence, \( i = 1, 2, \ldots, N \); where \( N \) is the total number of frames in the input sequence. Note \( i \) is also a time reference. We have \( K \) reference templates, labelled \( k = 1, 2, \ldots, K \); and each template has \( J(k) \) frames, labelled \( j = 1, 2, \ldots, J(k) \). We assume that we can calculate a frame distance measure \( d(i, j, k) \) to quantify the disparity between frame \( i \) of the input and frame \( j \) of the \( k \)th reference template. One possibility is a Euclidean distance between the parameter vectors. The \( k \)th plane of \( D(i, j, k) \) holds the cumulative distance measures for every reference template \( k \). The cumulative distance measures represent the sum of \( d(i, j, k) \) along the minimum distance path from grid point \((0, 0)\) to \((i, j)\).

The algorithm is given in figure 1.

```c
/* Initialise sentinels. */

for k = 1 .. N
    D(0, -1, k) = 0
    for j = 0 .. J(k)
        D(0, j, k) = infinity

for i = 1 .. N /* loop on test frames */
    for k = 1 .. K /* loop on reference templates */
        for j = 1 .. J(k) /* loop on frames in reference templates */
            D_prev = min { D(i-1, j-a, k) : 0 <= a <= 2 }
            D(i, j, k) = D_prev + d(i, j, k)

recognised = k : D(N, J(k), k) = min { D(N, J(t), t) : t = 1..K }
```

Figure 1 − DTW Isolated Word Recognition Algorithm

Note that we only allow the previous path to come from either the same frame, the preceding frame, or the previous frame but one \((a = 0, 1, 2, \text{ respectively})\). One extension of the above technique is to give each alternative \((a = 0, 1, 2)\) a probability weighting, which may be constant or may vary with \( i \) and \( j \).

Hidden Markov Models (HMMs)

HMMs aim to provide superior recognition by modelling each reference unit as a number of states and a state transition matrix. Each state has an output frame which is modelled probabilistically either with a discrete or continuous distribution. This uncertainty associated with a state output veils the identity of the generating state, hence the adjective ‘hidden’. The entries in the state transition matrix are the probabilities of a transition from one state to another in the reference unit. Transitions are usually constrained to be either a loop to the current state, a jump to the next state or a jump to the next state but one. These constraints are expressed as

January 26, 1989
zero probabilities in the state transition matrix for inadmissable state transitions.

HMM Recognition

The scheme is similar to DTW, but now the reference frames are the outputs of the HMM states. The HMM states are modelled probabilistically and so we need a statistical distance measure, such as a multi-variate gaussian distance metric.

When searching the grid the possible paths are weighted with the transition probabilities between reference states.

HMM Training

To train the HMMs, initial values are assigned to the model parameters and these are improved by an iterative estimation technique. Two grids of path distance measures are calculated for each word repetition in the training set against the current model. One is calculated moving forward across the grid, the other is calculated moving backwards. A weighted sum over the grids for all word repetitions is then used to re-estimate the model parameters. The model can be improved by repeating the process with the updated model parameters. It is possible to derive a distortion measure which indicates the probability of the new model generating the training set, and this can be used to control the number of iterations attempted.

Connected Word Models

In general we would like to recognise connected speech. It is possible to use word models trained from isolated words to recognise connected speech but in practice recognition performance is poor. Speakers run words together and this gives rise to co-articulation effects at word boundaries. Improved performance can be obtained by training the word models with data taken from connected speech.

A problem here is the segmentation of a training utterance into its constituent words. Manual segmentation is slow and error prone. One automatic solution is to use an isolated word model to recognise word boundaries in training utterances. These automatically extracted words can then be used to train a connected word model. To refine the results we repeat the process but now using the new model to perform the word boundary recognition. The connected word models can then be used in one of the recognition schemes described below.

Use Of Sub-Word Units

As vocabulary increases the number of models in a whole-word recognition system becomes unmanageable. Larger vocabularies can be accommodated by using sub-word units which are combined to form whole words. Various sizes of sub-units have been tried including demi-syllables and phonemes.

As with connected words, problems arise when attempting to segment a training database of connected utterances into sub-word units. Again, automatic routines can be developed which split utterances into elements of the
desired size.

**Connected Speech Recognition**

Usually we are interested in complete utterances, but connected methods also apply to the recognition of isolated words using sub-word units. With isolated word recognition we knew that each model had to start (end) at the beginning (end) of the input utterance, and that a single model would match the whole utterance. With connected recognition a model can start or end at any point in the input. Connected recognition schemes may attempt to find the optimal sequence of recognised units in terms of minimum cumulative distance. Alternatively, the connected speech recogniser could be used to section the input into recognised units rated with a confidence score, and delimited by start and end times. This will then provide the input to a more intelligent system which may use syntactic, semantic and pragmatic knowledge to provide more robust speech recognition or speech understanding.

Below we examine three classes of connected recognisers; one-pass, two-level, and level-building.

**One Pass**

The One Pass algorithm combines the reference unit matching and sequencing operations by extending the above DTW scheme to allow between-template transitions at template boundaries. For every input frame, $i$, we search for the template end-frame, $j = J(k)$, which has the lowest cumulative distance score and store the corresponding template number in an array $T(i)$. When we try to find a path to a start-frame, $j = 1$, we also consider the best end-frame in the preceding time slot. We need to store back pointers to allow the recognised sequence to be recovered at the end of the utterance. Array $B(i, j, k)$ holds the time index of the end frame of the previous matched unit in the current path. Note $T(B(i, j, k))$ gives the preceding template number. We use a further array, $F(i)$, to store the back pointers for the template numbers in $T(i)$. The algorithm is given in figure 2.

The One Pass algorithm is very efficient at finding the optimal sequence of units, but many good sub-optimal choices may be discarded early in the search.
/* Initialise sentinels. */

T(0) = 1
for k = 1 .. K
    D(0, 0, k) = infinity, B(0, 0, k) = 0
    D(0, 1, k) = 0, B(0, 1, k) = 0
for j = 2 .. J(k)
    D(0, j, k) = infinity, B(0, j, k) = 0

for i = 1 .. N /* loop on test frames */
for k = 1 .. K /* loop on reference templates */

/* Between-template transition rules. */

if (D(i-1, 1, k) > D(i-1, J(T(i-1)), T(i-1)))
    D(i, 1, k) = d(i, 1, k) + D(i-1, J(T(i-1)), T(i-1))
    B(i, 1, k) = i-1
else
    D(i, 1, k) = d(i, 1, k) + D(i-1, 1, k)
    B(i, 1, k) = B(i-1, 1, k)

/* Intra-template transition rules. */

for j = 2 .. J(k)
    j* = j' : D(i-1, j', k) = min { D(i-1, j-a, k) : 0<= a <=2 }
    D(i, j, k) = d(i, 1, k) + D(i-1, j*, k)
    B(i, j, k) = B(i, j*, k)
T(i) = t : D(i, J(T(i)), T(i)) = min { D(i, J(a), a) : a = 1 .. K }
F(i) = B(i, J(T(i)), T(i))

recognised sequence is obtained by;

r = N, R = 0 /* R is number of recognised units. */
while F(r) <> 0
    push T(r) /* Use a stack as we recover last unit first. */
    r = F(r) /* Follow back pointer. */
    R = R + 1 /* Increment number of units found. */

Sequence can now be pulled off the stack.

Figure 2 − One Pass Connected Recognition Algorithm

Two Level
The Two Level process is the most comprehensive scheme. For every pair of
start and end co-ordinates in the input utterance, b and e respectively (b < e),
we obtain a cumulative distance between every reference and the section of
the input starting at b and ending at e. We call this Do(b,e,k). The
next step is to find the best distance for any given b and e. We call this
D*(b,e) where;

D*(b,e) = min { Do(b,e,k) : k = 1 .. K }

January 26, 1989
In the second stage the best p-sequence of units, with \( p = 1..L \), has to be found. A p-sequence is a sequence of \( p \) reference units. If we let \( D^*(p,e) \) be the best match for a p-sequence ending at \( e \) then,

\[
D^*(p, e) = \min \{ D^*(b, e) + D^*(p-1, b-1) : 1 \leq b \leq e \}
\]

We can then find the optimum sequence length \( L^* \);

\[
L^* = L : D^*(L, N) = \min \{ D^*(L’, N) : L’ = 1 .. N \}
\]

The Two Level algorithm has the advantage that as well as finding the optimal sequence it is also possible to extract all the sub-optimal sequences. A major disadvantage of the Two Level algorithm is the large computational overhead.

**Level Building**

The Level Building algorithm is an attempt to reduce the computational complexity of the Two Level algorithm. Levels are labelled \( p = 1 .. L \), where \( L \) is the maximum number of levels. Each level corresponds to one recognised unit in the input utterance. The best matched unit in level \( p \) ending at frame \( i \) is labelled \( U(p,i) \).

At the first level, LB finds the most likely reference unit \( U(1,i) \) starting at frame \( 1 \) and ending at frame \( i \), for \( i = 1 .. N \). It will become the first unit in the decoded sequence if the first unit must end in frame \( i \). At the second level, for every unit \( U(1,i’) \) and for every frame \( i \), \( i > i’ \), the best second unit \( U(2,i) \) is found. More generally, at each level \( p = 1 .. L \), the best p-sequence ending at time \( i \) is computed.

A disadvantage of the LB algorithm is that it is necessary to fix some maximum value of \( L \), the expected number of units in the input utterance.

2. **Parallel Systems**

A convenient classification of hardware architectures is due to Flynn [2].

Conventional serial computers are classed as SISD (Single Instruction, Single Data) machines, where a single processor runs a program that manipulates a single data stream.

A MIMD (Multiple Instruction, Multiple Data) machine can be seen as a collection of SISD processors. Each processor can run a separate program on a separate data stream. A shared memory MIMD machine has a globally accessible block of memory which provides the communications medium between the multiple processors. A distributed memory MIMD architecture assigns a private block of memory to each processor, and processors communicate by message passing over some communications network. A transputer network is a distributed memory, MIMD machine with Inmos links providing communications between processing nodes.

January 26, 1989
MIMD machines are generally programmed in a task parallel manner where a number of concurrent tasks co-operate to solve a problem. Many languages for MIMD machines are based on a conventional programming language with extensions to provide task creation, synchronisation, and message passing. Examples include Parallel C for the transputer, and Lucid Lisp for the Intel IPSC/2. Other languages were designed from the outset for parallel systems, such as Occam implemented on the transputer.

A further class of machines have a single central control unit which broadcasts instructions to a large collection of simple processing elements which execute commands in lockstep. These are termed SIMD (Single Instruction, Multiple Data) machines. Examples include the ICL DAP, GEC GRID, the Connection Machine, and the PADMAVATI CAM. Typically arithmetic is carried out bit-serially but word-parallel. Each arithmetic instruction may take a comparatively long time to execute but the aggregate performance of the array is high. Also, for many applications only limited precision is required and then execution can be accelerated further.

The final class in Flynn’s classification is MISD (Multiple Instruction, Single Data). Here a number of processors execute different instructions on a single data stream. Processing pipelines are a common example of this arrangement which is analogous to a car production line. The degree of parallelism is proportional to the number of stages in the pipeline, which is limited by the number of clearly definable stages in the operation being performed. Floating point arithmetic is a popular candidate for MISD execution, with the vector processing units of the Cray series being a prime example.

Both SIMD and MISD machines exhibit data parallelism where the same operation is performed on a large set of data. Vectorising FORTRAN compilers have been developed which can find the embedded data parallelism in existing FORTRAN code to exploit data parallel execution, though always with a loss in efficiency compared to special purpose code. Extensions to conventional languages have been devised to allow data parallel operations to be performed efficiently on both SIMD and MISD machines. Examples include the FORTRAN-8X standard, and CM-LISP for the Connection Machine.

PADMAVATI

The PADMAVATI architecture has a hierarchy of hardware parallelism.

At the top level in the hierarchy, a Sun-3/260C acts as the host to a MIMD transputer network. The target configuration has 16 transputer nodes, each containing a T-800 transputer and 4-16 Mbytes of DRAM. Each transputer has four Inmos links providing high speed (20 Mbit/s) point to point communication between processor nodes. The links can be reconfigured into any topology using a crossbar switching network controlled by the Sun.

A lower level of parallelism is provided by the content addressable memory (CAM) boards which can be attached to each transputer node via an expansion bus interface. Each CAM board provides over 8000 SIMD associative processing elements, and up to 16 boards can be added to a node. The transputer acts as a host to the CAM array broadcasting instructions and data through control locations memory mapped into the transputer’s address space. Each
CAM cell has 36 bits of storage, a flag bit, and logic to implement a small but highly orthogonal instruction set including search, read, and write operations. The cells are arranged in a linear array and each cell can read its neighbours flag bit to allow inter-cell communication. Arithmetic operations are performed in a bit-serial, word-parallel manner using a scheme analogous to table look-up.

Within the PADMAVATI project Lisp and Prolog are being ported to the transputer. These are being extended with message passing primitives and accelerated by incorporating CAM into the language executor. In addition PADMAVATI is compatible with all standard transputer compilers, which include most major programming languages such as C, FORTRAN, and PASCAL. It is intended to support the use of CAM within these conventional languages by providing function libraries.

3. Parallelisation of the Speech Algorithms

The speech algorithms exhibit much potential for concurrent execution. Recognition and training have different computational requirements and so are considered separately.

Recognition

We would like to speed the recognition process both to attain real time recognition of continuous speech, and to reduce the time spent in developing new recognition algorithms.

The Two Level and Level Building algorithms as given above require the entire utterance to be input before processing can be completed. However [1] presents left to right versions of the algorithms where processing can step forward with each new input frame, ie. frame synchronously.

Much of the computation for each reference unit can be performed independently and hence concurrently. However it is necessary to broadcast each frame of the input and to perform global arbitration amongst matching words. This becomes more of a problem in a connected speech recogniser where word boundary information has to be collected and broadcast in every frame.

These communication overheads make the recognition process less amenable to parallel execution on loosely coupled systems such as distributed memory MIMD architectures. Shared memory MIMD architectures could suffer from contention problems when processors attempt to read globally accessible locations, but careful algorithm and system design could alleviate this. MISD pipelines can give a good but limited speed up for the arithmetic operations involved in recognition, but cannot easily be used to speed up the decision making and control processes.

SIMD architectures seem to match best the natural parallelism in the recognition process. SIMD machines support global broadcasting and decision making, and arithmetic can be performed efficiently in a data parallel manner.

Appendix A gives an algorithm for performing frame synchronous One Pass DTW connected speech recognition using the PADMAVATI CAM. Although the CAM was not designed for numerical applications recognition performance is still
faster than real time. This is partly due to the rapid parallel write and
global multiple response resolution typical of associative processors. It
should be straightforward to extend this to a Level Building scheme as this
can be viewed as several One Pass schemes operating in parallel [1].

A problem with SIMD is that it may involve significant effort to port an
existing sequential algorithm to a given processor array. This effort may
be justified when implementing a real time speech recogniser but not when
first developing and testing the recognition algorithms.

In general it is difficult to implement a sensible parallelisation scheme
for an algorithm which is still under development using conventional imperative languages. However in a research environment we will often want to assess a recognisers performance with a large database of input utterances. Each recognition run is independent and so can proceed in parallel using the farm paradigm described below for training.

Training

For speaker dependent systems training must be repeated for every new user, but for speaker independent systems this only has to be done once during development. The execution time becomes more important in a research environment where new models are developed and tested repeatedly.

The various forms of reference model have different training requirements. We consider here the training of connected unit HMMs with automatic segmentation of a training database as a complex and computationally demanding example for parallelisation.

We first note that when training a set of models, each model is usually trained in isolation from the others. A natural scheme for parallelisation is therefore to split the training into separate tasks for each model. A standard parallel programming paradigm which matches this form of task parallelism is the "task farm". A farm contains a master process and a number of identical worker processes. The master splits up the work into packets then farms out the packets to the worker processes. Each worker reads in a work packet, performs the desired task on the data in the work packet, then sends a result packet back to the master. For the training task we can place the database of input utterances with the initial unit model into the work packet, and return the trained model in the result packet.

For each model, we can process each utterance in parallel; performing segmentation, grid calculations and part of the final summation over the grid independently. We then combine the new partial sums to generate a new model, and iterate. Much of the processing within each utterance exhibits data parallelism, for example in the recognition process we repeat the local distance calculation for many points on the grid.

Extensive parallelisation schemes can involve a large software investment using the currently available explicitly parallel programming languages, but any physical parallel architecture has only a finite degree of parallelism. It is therefore desirable to choose the simplest parallelisation scheme which can efficiently utilise the physical parallelism of a given

January 26, 1989
architecture. We have noted that each reference unit can be trained separately in a processing farm. If the number of units to be trained exceeds the number of processors available then we can guarantee full utilisation of the available parallelism without having to write further algorithm specific code to extract the finer grain of parallelism within each training task. Fortunately the processor farm is one of the simplest parallel paradigms to implement, and one which can make most use of existing sequential software. We wrap the training routine in a wrapper which reads in work packets, calls the training routine, then formats and sends a results packet. We write a master routine to read in data, format and send work packets, and to collate the completed results packets.

Approximate figures for a training run are around 5-20 training tokens for isolated words, and around 100-1000 for sub-word units. These figures are then multiplied by the number of reference units to give the total number of training tasks in the run.

Farms on PADMAVATI

The 3L C system is being used to develop system software for PADMAVATI, and comes complete with a farm utility which allows the easy construction of parallel farms. The user simply supplies C source for the master and worker task and the system will then compile code which will automatically use any transputers attached to a network. Note that the master doesn't have to specify the destination of the work packet, the farm software handles routing and load balancing automatically. It should be possible to write generic master tasks and worker wrappers for speech work that would speed development of parallel code for new algorithms. As a performance guide, a single T-800-25 PADMAVATI transputer node (PARAPET) has run floating point intensive code around 6 times faster than the Sun-3/260C (with 68881 coprocessor). The transputer also supports rapid communications which can occur concurrently with integer and floating point operations, so the transmission of packets should not be a performance bottleneck.

Summary

Current speech algorithms have been discussed and shown to exhibit much potential for parallel execution. The different processing requirements of recognition and training have been shown to map efficiently onto SIMD and MIMD architectures respectively. An algorithm has been presented to perform One Pass connected speech recognition using the PADMAVATI CAM. Performance estimates show that this algorithm will run faster than real time on the PADMAVATI architecture. A scheme has been presented to perform training and recognition runs in parallel on a transputer array using the farm paradigm. The scheme requires only a minimal investment in new software and should give a near-linear speed-up with increasing numbers of transputers.

References

January 26, 1989
[1] Godin, C.; Lockwood P.  
"DTW Schemes for Continuous Speech Recognition: a Unified View"  
to be published?  
Flynn, M.J.  
IEEE Trans. Computers C-21, 9 (September 1972), 948-960
Appendix A

Use of PADMAVATI CAM for Continuous Speech Recognition

As an example of an SIMD machine performing recognition, and to show the
potential of the PADMAVATI CAM for speech processing an algorithm is
described for performing One Pass DTW connected recognition. The CAM algo-
rithm is frame synchronous and has an execution time independent of the
number of reference templates. First a description of an associative
dialect of ‘C’ is given then the recognition algorithm is presented in this
associative ‘C’.

Associative ‘C’

Associative ‘C’ is an attempt at defining an efficient high level language
for expressing the associative data-parallel operations supported by the
CAM.

The format of the items processed in the CAM is declared in a similar manner
to conventional C structure declarations. This is consistent with the view
of the CAM as active memory of the host transputer. The "union" and
"struct" definitions have been extended to work at the bit field level
rather than at machine word level. "Union"s indicate where bit fields which
are not required simultaneously can use the same storage. Note that the
same structure definition can be used in RAM or CAM, but the CAM has 35-bit
wide data words.

The CAM doesn’t support conventional RAM co-ordinate addressing and so all
data structures must be accessed associatively. Array indices and structure
member names are equivalent means of specifying sub-portions of a large data
structure and can collectively be termed tags. To allow rapid access to
portions of a structure in associative memory the values of these tags must
be stored explicitly along with the data items. A trade off between storage
space and access speed is possible by only storing a tag once for each
dimension or structure nesting depth and then using the CAM’s block marking
facilities to restrain further searches to the particular sub-structure or
array sub-space. Conventional RAM accesses to array elements with variable
indices requires run time calculations involving a multiplication per dimen-
sion, and so the associative approach compares well in run-time efficiency.
One advantage of the associative array indexing method is that only the ele-
ments which are used in a large array need to be physically represented,
thus facilitating the rapid manipulation of large sparse matrices. Also
multiple elements can be simultaneously specified by allowing "don’t care"
index values enabling data parallel operations to be expressed cleanly.

In assignment operations the l.h.s. of the assignment is evaluated to gen-
erate an "l-value" which indicates where the value of the r.h.s expression
is to be stored. A data item in RAM is addressed by an l-value which is a
single pointer and this uniquely identifies a single storage cell. When a
data item is in CAM, the meaning of an l-value is translated to be a
sequence of search operations to locate the data field. Several CAM loca-
tions may match this sequence of search operations and so the l-value no

February 3, 1989
longer identifies a single storage location. The semantics which follow naturally are that all matching locations should be written to in parallel. The CAM hardware implements this parallel write operation in a single access cycle offering a large speed up over conventional implementations of this operation.

The r.h.s. of an assignment can similarly evaluate to a CAM r-value. This r-value is a set of operations to extract the appropriate fields. Again multiple fields can be specified. If the l.h.s. of the assignment is a CAM l-value then a parallel shift operation is set up in the CAM. If the l.h.s. evaluates to a conventional l-value then this must be an array of the correct dimension to receive the specified values.

These parallel operations are made available in associative ’C’ by the introduction of pattern matching indices and relative indices. Indices of CAM arrays are considered as identifying tags of the array elements and pattern matching is supported with a star "*" signifying "any" subscript in this position. Relative indices are used to show parallel shifts of data in an array, and these are represented syntactically as double square brackets. Thus the parallel assignment "fred[*].one = fred[[x]].one" moves in parallel the values of all "one" fields in the structures in array "fred" into the "one" fields of structures "x" steps along. In structures there are no variable tags, elements are always referred to by constant names, but the equivalent operations are supported, eg. "sheila[*].fred = sheila[*].jim" moves all the values of "fred" in the array of structures into the "jim" fields. These two constructs can be combined to allow say "janine[**][**].sheila=janine[[-2]][[-1]].sam".

The "with" construct is introduced from PASCAL to ease the writing of code given that scarcity of memory space in the CAM forces greater use of "union"s than is normal in C. The "with" allows partial specification of structure or array names within the construct. The compiler may also optimise the operations within the "with" statement.

The "select" construct allows further specification of data sub-sets. It’s argument is a logical conjunction of equalities, inequalities, max and min operations applied to a data structure. Within the "select" only those data items matching the conditions in the "select" argument are operated on. The same partial specification of data item names is supported as provided in the "with" statement. "select"s can be nested, with inner selects implicitly dependent on outer "select"’s search conditions.

It is useful to allow the programmer to read back the value of array indices of elements which match some criterion such as a "select". To enable this, it is possible when declaring arrays of CAM structures to attach optional subscript labels for each dimension, eg "CAMstatic int R[1000:i][20:j]:16;". These subscript labels can then be used as structure names to extract the indices as data fields (which is in fact how they’re stored). For example we may write,

```
select (min(R[*][*]))
{
  min_i = i;
  min_j = j;
}
```

February 3, 1989
to find the position of the minimum value in the previous array. These
index labels remain in scope for all lower dimensions and sub-structures so
cannot be used as sub-structure names within the same array.

Enhanced type checking is used to control the generation of code to perform
bit-serial CAM arithmetic to the necessary precision.

Connected Recognition Algorithm

The following algorithm illustrates the use of associative 'C' and shows how
the CAM can be used for continuous speech recognition.

typedef unsigned Nat; /* Shorthand for positive or Natural numbers. */

/* All the following word formats have left enough room for the array */
/* indices and sub-structure tags to be stored, these would be */
/* allocated by the compiler. */

/* Format of word used to store reference vector component, and in */
/* which local distance calculation is performed. */

typedef struct
{   union /* Workspace to calculate local distance measure. */
    { struct /* Workspace for first stage. */
        {   union
            { int param : 7; /* Input frame parameter. */
                int diff : 7; /* (input-ref). */
            } in;
            Nat sq : 14;  /* (input-ref)^2. */
        } one;
        Nat sum : 18;  /* Holds sum of (input-ref)^2. */
    } ws;
    int ref : 7;  /* Reference frame parameter. */
} Param; /* Held in one CAM word. */

/* Format of words in which cumulated distance is stored and compared */
/* against previous frame values and/or the previous best end frame. */

typedef struct
{   Nat d : 21; /* Cumulative distance. */
    Nat bp : 9; /* Back pointer. */
} Dist; /* Held in one CAM word. */
/* Format of word used to store bits to select sub-sets of frames. */

typedef struct
{   Nat eot : 1;  /* Tags this frame as "end-of-template" to enable */  /* templates to use less than the maximum number of */  /* frames, even though these must still be allocated */  /* to give a regular array to allow parallel CAM */  /* data shifting. */  
    Nat it1 : 1;  /* Set to one for frames j who need to look at frames */  /* j-1 in inter-template transition function. */  
    Nat it2 : 1;  /* Set to one for frames j who need to look at frames */  /* j-2 in inter-template transition function. */  /* (Note it2 => it1). */  
} Flags;

#define FRAME_LEN    (16) /* Number of parameters in a frame.*/

typedef struct
{   Flags f;
    Dist h[2:a];  /* Two distance words. */
    Param parms[FRAME_LEN:p]; /* Reference frame vector. */
} Frame; /* Needs 19 CAM words for 16 parameters. */

#define MAX_FRAMES   (25)     /* Maximum number of frames in a template. */
/* Use "j" to label frame index. */

typedef Frame Template[MAX_FRAMES:j];

#define MAX_TEMPS    (270)    /* Maximum number of templates.*/
/* Declare array of reference templates in CAM memory. Use "k" to label */
/* template index. */

CAMstatic Template R[MAX_TEMPS:k];
#define MAX_I  ( /* Number of input frames. */ )

/**************************************************************************
/* These arrays are held in RAM on the host transputer. */
/**************************************************************************/

int references[MAX_TEMPS][MAX_FRAMES][FRAME_LEN];  /* Reference templates.*/
Nat J[MAX_TEMPS];                                      /* Frames per template. */
int input[ MAX_I ][ FRAME_LEN ];                      /* Input sequence. */
Nat T[ MAX_I ];                                        /* Holds template number with best end */
                                        /* frame match at time i. */
Nat Dist[ MAX_I ];                                     /* Holds D( i, J(T(i)), T(i) ). */
Nat F[ MAX_I ];                                        /* Holds B( i, J(T(i)), T(i) ). */

 initialise()
 {  /* Load CAM with reference templates and clear cumulated distances and */  /* back pointers. */
    Nat k, j, p;
    Nat k, j, p;
    R[ ][ ].f.eot =0;
for(k=0; k < MAX_TEMPS; k++)
        with (R[k])
            {  for(j=0; j < J[k]; j++)
                        with([j])
                            {   for(p=0; p < FRAME_LEN; p++)
                                parms[p].ref = references[k][j][p];
                                if (j == J[k] - 1)
                                    f.eot = 1;
                            }
            }
    R[ ][ ].f.it1 = 1;
    R[ ][ ].f.it2 = 1;
    R[ ][0].f.it1 = 0;   /* Zeroth frame doesn’t do intra-template */
    R[ ][0].f.it2 = 0;   /* transition functions. */
    R[ ][1].f.it2 = 0;   /* First frame only does -1 transition. */
    R[ ][0].h[0].d = MAX_INT;  /* Set all cumulative distances to */
    R[ ][0].h[0].bp = 0;      /* maximum, and clear back pointers. */
    R[ ][0].h[0].d = 0;     /* Reset first frames to have zero distance. */
}

February 3, 1989
find_prev_min(i)
{
    /* This performs the decision function for both */
    /* between template and inter-template transitions and stores */
    /* the best previous cumulative distance in "R[*][*].h[0].d" */
    /* for every frame, together with the appropriate back pointer */
    /* in "R[*][*].h[0].bp". */
    /* 323 + 2190 = 2513 Cops. */
    /* 323 + 2190 = 2513 Cops. */
    /* Between-template transitions, 4 + 319 = 323 Cops. */
    with (R[*][0]) /* First frame in every template. */
    {
        h[1].d = Dist[i-1]; /* Broadcast best previous minimum = 4 Cops. */
        select ( h[0].d > h[1].d )
            {
                h[0].d = h[1].d;
                h[0].bp = i - 1;
            } /* 315 + 4 = 319 Cops. */
    }
    /* Intra-template transitions. 2*(546 + 549) = 2190 Cops. */
    with (R[*]) /* For every template. */
    {
        select ( [*].it1 == 1 )
            {
                h[1].d = [[-1]].h[0].d; /* Copy previous frame*/
                /* distance forward = 546 Cops. */
                select (h[1].d < h[0].d)
                    {
                        h[0].d = h[1].d;
                        h[0].bp = [[-1]].h[0].bp;
                    }; /* 315 + 234 = 549 Cops. */
            }
        select ( [*].it2 == 1 )
            {
                h[1].d = [[-1]].h[0].d; /* Copy best of [[-1]] */
                /* and [[-2]] forward = 546 Cops. */
                select (h[1].d < h[0].d)
                    {
                        h[0].d = h[1].d;
                        h[0].bp = [[-1]].h[0].bp;
                    }; /* 315 + 234 = 549 Cops. */
            }
    }
}
sum(array)
{
    /* This code sums the components of a CAM vector storing the sum in */
    /* the first member of the vector. The code uses a binary tree of */
    /* additions. In the first cycle all odd index words are added to */
    /* all even index words and the result is stored in even index */
    /* words. If we now consider the reduced array of results we can */
    /* repeat the process reducing the number of partial sums by two */
    /* every time. We find that to reduce over addition a vector of */
    /* n = 2^N k-bit words spaced s words apart initially requires, */
    /* */
    /* I = s*(2^N*(N+k-2)+2-k)+N*(7N+14k-8) CAM instructions */
    /* */
    /* compared with K*(2^N-1) instructions (K some constant) for a */
    /* sequential machine. However the CAM can perform many such */
    /* reductions in parallel and the CAM instruction count includes */
    /* all timing for operand access. In this example one such */
    /* reduction is performed simultaneously for every frame of every */
    /* template. */
    /* */
    /* The code isn't shown here for brevity. */
    /* */
    /* For this example this routine takes 1108 Cops. */
}

add_local_distance(i)
{
    /* This adds the local distance measure to the best previous */
    /* cumulative distance measure held in "h[0].d". */
    /* */
    /* 1969+18 = 1987 Cops. */

    Nat p;

    /* Broadcast input frame = 18 Cops. */

    for ( p = 0; p < FRAME_LEN; p++ )
        R[*][*].parms[p].ws.one.in.param = input[i][p];

    /* Do calculation. 496 + 1108 + 365 = 1969 Cops. */

    with (R[*][*]) /* For every frame in every template. */
    {
        with (parms[*]) /* For every parameter word in every frame. */
        {
            ws.one.in.diff = ws.one.in.input - ref;
            ws.one.sq = ws.one.in.diff * ws.one.in.diff;
        } /* 62 + 434 = 496 Cops. */

        sum(); /* Sum using binary addition tree = 1108 Cops. */
        h[0] = h[0] + parms[0].ws.sum; /* Rounded up = 365 Cops. */
    }
}

February 3, 1989
get_best_end(i)
{
/* This finds the template with the best matching end frame and */
/* extracts values for T[i], Dist[i] and F[i]. */
with ( R[*][*] ) /* For all frames of all templates. */
{ select (label.eot == 1 && min( h[0].d ) )
   { T[i] = k; /* Retrieve it’s template number, */
     Dist[i] = h[0].d; /* cumulative distance, */
     F[i] = h[0].bp; /* and back pointer. */
   }
} /* 63 + 3 + 4 = 70 Cops. */
}

recognise()
{
Nat i;
for ( i = 0; i < MAX_I; i++ )
{ /* Once round the recognition loop takes 2513+1987+70 = 4570 Cops. */
   find_prev_min(i); /* 2513 Cops. */
   add_local_distance(i); /* 1987 Cops. */
   get_best_end(i); /* 70 Cops. */
}
}

main()
{
   initialise();
   recognise();
}

Further work is necessary to determine the minimum precisions required at
various stages in the processing to maintain good performance. In this
example the input and reference parameters are represented as 7-bit 2s com-
plement fixed point numbers. A Euclidean distance metric is calculated with
full precision to yield an 18-bit result. The 9-bit back pointers allow up
to 512 frames in an input sequence, corresponding to over 5 seconds of unin-
terrupted speech at 10ms per frame. With a sequence of 512 frames we could
get a cumulated distance requiring 18 + 9 = 27 bits of precision. We reduce
the precision to 21 bits by losing some LSBs and having an overflow bit
instead of the MSBs. If the cumulated distance overflows then it is not
likely to be part of a succesful match. The exact precision reduction
scheme adopted can be determined by experimentation.

February 3, 1989
Performance Estimates

The algorithm is only a first attempt at performing connected speech recognition but is realistic in the amount of computation performed. The associative C used to describe the algorithm has not been implemented but is used here to save presenting pages of obscure CAM instruction codes. Library routines which perform the same operations as used in the above algorithm have been written and tested on a functional simulation and so we can derive timing figures for the recognition algorithm. The "Cops" figures annotating the listing are the number of CAM operations required to hand code the functions described in associative 'C'.

We find that for every frame of input we need to perform 4570 CAM instructions. Apart from reading the best end frame value, all operations are formed from writes and searches. On the PADMAVATI architecture we can use the transputer block move instructions to perform the role of a microsequencer reading out a stored sequence of write/search instructions from transputer RAM. One instruction executed in this manner requires two RAM fetches (one instruction, and one data) and two bus transfers. There is no transputer instruction fetch overhead because the block move is executed by transputer micro-code. With a 25 MHz T-800 a RAM fetch takes 200ns on PARAPET and a bus access takes approx 500ns, hence a single instruction will take around 2*(200+500)=1400ns. The transputer has to execute some code to extract and broadcast the best end-frame but the time taken for this is negligible. Hence the estimated time to perform recognition calculations per frame is around 6.4ms, which is faster than real time if we assume a 10ms frame rate. The algorithm could be further optimised for the CAM architecture, and if the transputer interface was replaced with a special purpose controller then the CAM cycle time could be reduced to around 200-250ns per instruction. The effect of these speed-up procedures would be to allow more processing per frame to enhance recognition performance.

Each frame occupies 19 CAM words, and each reference template can have up to around 20-25 states, giving a total of 475 CAM words per template. Each board holds about 8000 CAM words and so a 16 board CAM array could process around 270 reference templates simultaneously. During the calculation of the distance metric the CAM performance peaks at 1.2 GOPS for the 7-bit subtractions, 178 MOPS for the 7-bit to 14-bit squaring operation, and 65 MOPS when performing the summation of 14-bit numbers to give an 18-bit distance metric. Special purpose hardware could outperform the CAM in both space and time on this application, but the CAM has the advantage of being a general purpose computing structure.

February 3, 1989