Spoken Natural Language Understanding as a Parallel Application

K. Asanovic, J. R. Chapman

GEC Hirst Research Centre, East Lane, Wembley,
Middlesex HA9 7PP, United Kingdom
{Electronic mail krste@uk.co.gec-rl-hrc}

Abstract. A parallel machine, consisting of a network of transputers with content addressable memory (CAM), a rapid global communications network and runtime support for Lisp and Prolog is presented. A natural language understanding system based on the blackboard model is described, and shown to exhibit parallelism at a number of levels. A suitable topology for the system is presented, and it is shown how this is mapped onto the transputer network. The use of the CAM to accelerate various natural language processing algorithms is discussed.

Introduction

The GEC Hirst Research Centre is developing a speech understanding system as an example of an application for a parallel architecture. The first section of the paper describes the hardware architecture, the system software and the supported programming languages. The second section describes a sequential implementation of the speech understanding software. The final section describes how the application software is adapted to the parallel architecture.

Keywords

Transputer, Content Addressable Memory, Parallel Architecture, Parallel Lisp, Speech Understanding, Natural Language Processing, Blackboard Model

1. System Architecture

The architecture is based on PADMAVATI, which was designed in collaboration with Thomson-CSF and CSELT as part of the Esprit programme [1].

1.1. Hardware

The hardware consists of sixteen transputer-based nodes with a rapid global communications network and substantial quantities of content addressable memory (CAM). A Sun-3/260C acts as host to the network.

Each node occupies a single double-size Eurocard and contains a 20MHz T-800 transputer, 8-16 Mbytes of dynamic RAM and an expansion bus interface. The expansion bus is used to attach CAM boards and other memory or peripheral boards to the node.

The CAM is an original design and is formed from custom VLSI chips connected in a linear array. Each VLSI CAM chip contains 148 36-bit data words and supports a small number of orthogonal instructions. The CAM boards can be cascaded to form a larger CAM array, for example 16 boards form a single linear array of 127872 words. A CAM array can be allocated in software to form multiple independent banks.

A single 20Mbit/s link from each node is connected to a multi-stage packet-switched delta network to provide rapid global communication. The design allows incremental expansion up to 256 nodes. This expansion retains full connectivity and increases available communications bandwidth, yet still requires only one link per node. Link adaptors convert between the bit-serial protocol of the links to the byte-parallel protocol of the stages in the delta network. The VLSI switching elements are intelligent and provide buffers, routing and contention handling to maximise use of the available parallel communications bandwidth. In addition cut-through or ‘worm-hole’ routing is supported, giving an achievable minimum message latency of 1.4 us for a 16 node network.

One node is designated the communications processor (CP) and has a link reserved to provide the interface to the host Sun workstation. The remaining links can be reconfigured to provide fast local communication in a topology tuned to a particular application.

1.2. ARTiSt

ARTiSt (A Run TIme SysTem) is a ‘software backplane’ similar in approach to the distributed operating systems V [2], Amoeba [3] and Helios [4], and provides distributed runtime support for multiple tasks on a transputer network. Each node runs an instance of the ARTiSt kernel which provides topology transparent communication primitives.

Tasks communicate via variable length messages sent through the ARTiSt kernels. Long messages are split into packets for transmission. All inter-node communication makes maximal use of the transputer’s link engines to off-load the CPU. ARTiSt carries four different types of message labelled SIGNAL, REQUEST, REPLY and MAIL. SIGNAL messages are used to implement software interrupts and exception handling.

REQUEST and REPLY messages form the two ends of a synchronous communication. These are intended for communication between system server and user client tasks. A user client task sends a REQUEST to a system server task then blocks waiting for a REPLY from the server. MAIL messages implement asynchronous communication. The sender task continues execution after the MAIL operation. SIGNAL messages are queued separately and appear on the signal channel of a task. Arriving REQUEST and MAIL messages are buffered in a single queue for each task. A task can ask to receive the first message from any source or from a particular source. REPLYs do not need to be buffered as the receiving process will be blocked waiting for the REPLY. REPLYs effectively jump the queue of waiting messages.

All conventional operating system services are provided by system server tasks. Each server task has an associated library of routines which implements a procedural interface to hide the client-to-server message protocol. The Unix file server task provides access to the Unix file system of the host. As Unix maps all devices into the file system, all host I/O can be accomplished through the file server. Each task is assigned an independent set of file channels. Both UFS and the server running on the host are multitasking and allow multiple I/O requests to be in progress at any time.

1.3. Parallel Lisp

Initially Le_Lisp is to be the Lisp dialect supported by PADMAVATI. All Lisp I/O makes use of the procedural library supplied by the UFS which hides the distributed nature of ARTiSt on PADMAVATI. Primitives are added to support asynchronous inter-task communication through the use of ARTiSt MAIL messages. The MAIL primitive is non-blocking and sends a MAIL message to a named task. The RECEIVE primitive has arguments for the source and type of message to be received and can be either blocking or non-blocking.

When writing a parallel Lisp application for PADMAVATI, the user generates separate Le_Lisp source for each process. The user also supplies configuration information. This gives task names, task source file-names, a system topology and a mapping of tasks to processors. A configuration routine uses this information to compile ARTiSt code which is then linked with user code to form a single boot file for the network. All Lisp processes on a node share code forming the Le_Lisp base system.

Investigations are continuing into the potential for accelerating Lisp language execution with the CAM. Possibilities being considered include maintaining variable bindings and/or association lists in CAM. As a first step, Lisp primitives are added to allow the user direct control of the CAM.

Other languages being developed for PADMAVATI include macro- and micro-parallel versions of Prolog extended with ARTiSt compatible communication primitives.

2. Natural Language Application

The application software is being written as part of the Esprit project Ikaros [5], in collaboration with CGE and the University of Stuttgart. The software is written in Lisp and derives a database query from a lattice of basic phonetic units supplied by an acoustic phonetic decoder.

2.1. Ikaros Blackboard

The blackboard model [6,7] is adopted with all information stored on the blackboard in the form of hypotheses. A typical hypothesis might be "starting at time-frame 17 and ending at time-frame 78 is a noun phrase", plus various associated information. The blackboard acts as a global shared memory for a collection of cooperating linguistic knowledge sources (KSs). KSs are triggered by new hypotheses appearing on the blackboard and will then themselves generate further new hypotheses.

Figure 1: The Blackboard

The blackboard is partitioned into levels. Typically a KS will find its input at one level, and put its output onto another level. Some KSs operate in a "top-down" way, and others operate "bottom-up". The interplay of these two forms of processing enables "hole-filling" to take place; words are detected bottom-up in regions of the utterance where the acoustic phonetic decoder has generated accurate phoneme hypotheses, and top-down where the phonemes have not been so accurately recognised. Figure 1 shows the overall structure of the system.

2.2. Blackboard and Knowledge Source Interaction

In the Ikaros blackboard system, there are three sorts of interaction between KSs and the blackboard. These are: KS triggering, KS requests for hypothesis information and writing of KS results on the blackboard.

When a hypothesis is written on the blackboard, various KSs can be triggered by that hypothesis. The blackboard knows which levels are of interest to which KSs. For example, the boundary-based word finder KS is triggered by hypotheses at checked_word level. A typical cycle of execution might involve this KS being triggered by the hypothesis "The word Birmingham appears, starting at time 112 and ending at time 176, with confidence 28%".

While processing a hypothesis, a KS will need to interrogate the blackboard for information about other hypotheses. For instance, the above word-finder example will ask the blackboard for details of the phoneme-level hypotheses starting from time coordinate 176, the end time of "Birmingham".

The output of a KS is further hypotheses to be written on the blackboard. The word-finder KS above could eventually produce a hypothesis "the word ‘train’ appears starting at coordinate 176 ..." The KS informs the blackboard at which level the hypothesis is to be written.

2.3. Sequential Control Strategy

The blackboard model employs a number of independent KSs, and a blackboard monitor and scheduler. The scheduler has to determine which KS to execute next and which of the outstanding hypotheses that KS will process in the next cycle. Local and global heuristics operate on the expected efficiency of the KS, the confidence score of the data and other information, to yield an execution schedule.

3. Parallelisation of the Application

The application exhibits much potential for relatively coarse-grained parallelisation on a distributed memory, message-passing architecture such as PADMAVATI.

3.1. System Topology

The simplest parallel topology has a central blackboard task with satellite KS tasks. Although the blackboard is not computationally intensive, a central blackboard task would form a communications bottleneck.

The blackboard is conceptually a shared global memory, but typically KSs will only interact with one or two of the blackboard levels. It therefore seems natural to split the blackboard, with separate tasks for each level.

The original blackboard was responsible for maintaining prioritised work queues for each KS. For further parallelism, it is possible to have a separate KS queue (KSQ) task for each KS. This receives hypothesis triggers from blackboard levels and maintains them in priority order until the KS requests the next task.

All these parallelisation measures have involved placing distinct tasks into separate concurrent tasks. A finer grain of parallelism is possible within the word-finder KSs. These carry out very CPU-intensive tasks which search the phoneme lattice and the phonetic lexicon, generating possible words. The phoneme level is written to the network from the acoustic phonetic decoder and is never altered. It is therefore possible to provide multiple word-finder tasks operating on copies of the phoneme lattice. Each word finder is assigned a separate portion of the lexicon, and searches the lattice only for words in that portion of the lexicon. Other KSs need to read from the phoneme level, and as there are multiple copies present, the work involved in servicing these requests can be shared.

With this scheme so far we have 10 blackboard level tasks, 15 KS tasks and 15 KSQ tasks, with the possibility of further word-finder KS tasks and phoneme level blackboard tasks. Figure 2 indicates a possible mapping of tasks to the 15 free processors in PADMAVATI (the 16th processor is the CP). We group KSQ tasks together with their corresponding KS, on the basis that KSQs consume little CPU time and perform most communication with their KS.

Figure 2: Division of KSs and levels between 15 transputers

3.2. Blackboard Level Tasks

Each blackboard level task (BBLT) reads in MAIL messages as they arrive. Messages are either instructions to write to the blackboard level or requests for information about hypotheses. When a new hypothesis is written, the BBLT MAILs trigger messages to the appropriate KSQs. When a request for information arrives, the BBLT searches the blackboard for the required information and MAILs the data back to the requesting KS.

3.3. KS Tasks

The KSs MAIL their KSQ asking for the next task, then perform a blocking RECEIVE waiting for the reply. When a task arrives, the KS resumes processing. During the course of processing, the KS will MAIL requests for information to BBLTs and will block, pending replies. As the KS generates new hypotheses, these are MAILed asynchronously to the appropriate BBLT.

3.4. KSQ Tasks

These accept either trigger messages from the BBLT or requests for tasks from the KS. As triggers arrive, they are inserted in priority order into the KS work queue. When a KS requests a task, the highest priority task can be returned immediately if there are any tasks in the queue. Otherwise the KSQ will wait until a task arrives, then forward that directly on to the KS.

3.5. Use of Content-Addressable Memory

Many of the natural language processing algorithms used by the tasks can be accelerated with the CAM.

For example, the nucleus-based word-finder KS needs to detect words, even though the basic phonetic data is inaccurate. It might, for instance, detect the word "Birmingham" given the phonetic information: "Stop, Vowel, M, I, Nasal, A, M". If the lexicon is stored in the CAM, words can be searched for associatively, with "don’t care" bits set for phonetic features of low confidence. Any matches can then be retreived and used to generate word hypotheses.

Parsing can be greatly accelerated if partial results (i.e. the parser’s chart or well-formed substring table) are stored in the CAM.

The CAM is also useful for blackboard access. A KS can ask the blackboard to list all the hypotheses with some property or combination of properties. For example, a KS might ask for a list of all the hypotheses whose start coordinate was 117 (this is a slight oversimplification since the parser, for example, will make use of more information than just time coordinates in deciding whether two hypotheses might be adjacent). The present application software stores such information using association lists;

the content addressable memory provides an ideal mechanism to support this more efficiently.

4. Conclusions

A spoken natural language understanding application has been demonstrated as exhibiting considerable parallelism at a number of levels, and an architecture has been presented which is capable of realising such parallelism. Coarse-grained parallelism is observed with concurrent knowledge sources and blackboard levels. Finer-grained parallelism has been identified within the word-finding tasks, which can be split between separate processors. Finally, certain associative operations within various natural language processing algorithms can be accelerated by the fine-grained parallel hardware of the content addressable memory.

Acknowledgements

This work was partially funded by Esprit projects 1219 and 954.

References

[1] Esprit Project 1219, PADMAVATI (Parallel Associative Development Machine As a Vehicle for ArTificial Intelligence) "Deliverable on System Definition 1".

[2] D.R. Cheriton, "The V Distributed System" CACM 31: 314-333, 1988.

[3] A.S. Tanenbaum, R. van Renesse, "Distributed Operating Systems" ACM Computing Surveys 17: 419-470, 1985.

[4] N. Garnett, T. King, J. King, "Helios Developers Notes", Perihelion Software (Version 1.0, 1987).

[5] Esprit Project 954, IKAROS (Intelligence and Knowledge Aided Recognition of Speech), Press Release, 1987.

[6] P. Nii, "Blackboard Systems: The Blackboard Model of Problem Solving and the Evolution of Blackboard Architectures" The AI Magazine, Summer 1986, 38-53.

[7] D.L. Erman, F. Hayes-Roth, V.R. Lesser, D.Raj. Reddy, "The HEARSAY-II speech understanding system: Integrating knowledge to resolve uncertainty." ACM Computing Survey 12: 213-253.