To appear in the November 1994 issue of IEEE Computer: Special Issue on Associative Processing.

ASC - AN ASSOCIATIVE COMPUTING PARADIGM

by

Jerry Potter
potter@mcs.kent.edu
(216) 672-4004 x264

Johnnie Baker
jbaker@mcs.kent.edu
(216) 672-4004 x260

Arvind Bansal
arvind@mcs.kent.edu
(216) 672-4004 x214

Stephen Scott
sscott@mcs.kent.edu
(216) 672-4004 x215

Chokchai Leangsuksun
chokchai@mcs.kent.edu
(216) 672-4004 x215

Chandra Asthagiri[1]
asthagir@mcs.kent.edu
(216) 672-4004 x264

Kent State University, Kent, Ohio 44242
ftp://ftp.mcs.kent.edu/asc

Abstract

The associative computing (ASC) paradigm provides data parallel programming and constant time operations for searching, maximum, and minimum. A model of computation for ASC is given which supports algorithm development and analysis. The single instruction stream version of the model can be supported by current SIMD computers and it is feasible to build a computer that supports multiple instruction streams, allowing a more efficient utilization of the processors. Data representation techniques and an application of ASC to both graph theory and logic programming are included. This paradigm combines numerical and non-numerical computing in a general and robust way, making it suitable for a wide range of applications.

Keywords: *SIMD computer, structure codes, massively parallel, associative computing, multitasking SIMD, computational models, data parallel programming*

INTRODUCTION

This paper describes the ASsociative Computing (ASC) paradigm which is the result of multiple research projects at Kent State University. One goal of this research is to develop a parallel programming paradigm that is suitable for a wide range of applications, efficient to write and exe-

1. Department of Computer Science and Engineering, Wright State University, Dayton, Ohio 45435.-

cute and that can be used on a wide range of computing engines from PCs and workstations to massively parallel super computers. The ASC paradigm uses massively parallel search by content and processing of unordered data. This approach to associative computing originated with the bit serial SIMDs[2] of the 1970's when computers were expensive and unordered searching was cost effective only on massively parallel bit serial machines. The continuing trend toward lower hardware prices and higher software costs suggests that some of the increase in hardware productivity should be used to improve programming productivity by using associative techniques on conventional sequential and parallel machines.

ASC uses two dimensional tables as the basic data structure. It has an efficient associative based, dynamic memory allocation mechanism which does not use pointers. ASC incorporates data parallelism at the base level, relieving programmers of the responsibility of specifying low level sequential tasks such as sorting, looping, and parallelization.

ASC supports all of the standard data parallel and massively parallel computing algorithms. It combines numerical computation such as convolution, matrix multiplication, and graphics with non-numerical computing such as compilation, graph algorithms, rule based systems, and language interpreters[10]. The KSU research and this paper concentrate on non-numerical aspects of ASC.

The remainder of this paper is organized as follows: the next section defines the ASC model. The third section describes characteristic ASC techniques such as constant time operations, tabular data structures, responder processing and control parallelism. The fourth section discusses two representative applications. The fifth section discusses implementation issues and the current research in extending the ASC paradigm to new areas. The last section is a summary. Many of the referenced papers and related technical reports can be obtained by anonymous ftp from ftp.mcs.kent.edu then change directory to asc.

THE ASC MODEL

The ASC model is the basis of a high level associative programming paradigm and language. It is an evolution of the general associative processing techniques developed in the early 1970's by Goodyear Aerospace with the STARAN SIMD computer[2]. The extended model described in Figure 1 is intended to provide a basis for algorithm development and analysis similar to the PRAM models with the additional provisions that hardware can be built to support this model and that the primitive operations of the model are sufficiently rich to allow efficient utilization of massive parallelism[14]. These features allow parallel algorithms to be developed for large problems which then can be abstractly analyzed and executed. Furthermore, algorithms based on a common model will have greater applicability and retain their importance longer than if based on a specific computer that may be out of production within a few years. Briefly, the model calls for data parallel execution of instructions, constant time associative searching, constant time maximum and minimum operations, and synchronization of instruction streams using control parallelism. Figure 1 lists the specific properties that the hardware must have to support the model.

Reflecting the specifications in Figure 1, ASC is a new associative computing paradigm and language characterized by built-in associative reduction notation, associative responder iteration, responder based control of flow, responder reference and selection mechanisms and a multiple instruction stream capability that provides dynamic control parallelism on top of data parallelism. ASC supports recursion and special command constructs with automatic backtracking for complex context sensitive searching. Fundamental to the non-numerical focus of ASC are the unique structure code features and dynamic memory allocation. ASC contains many other minor new features such as the sibdex function and pronouns. The most important features of ASC are discussed in the next section. ASC language syntax is described in detail in[10].

1. CELL PROPERTIES:
   a. The memory of an associative computer is composed of an array of cells. Each cell consists of a processing element (PE) and a local memory. See Figure 2.
   b. There is no shared memory between cells. Each PE can only access the memory in its own cell. The cell's PEs may be interconnected with a modest network (e.g., a grid).
   c. Related data items are grouped together (associated) into records and stored typically one per cell. It is assumed that there are more cells than data.
2. INSTRUCTION STREAM (IS) PROPERTIES:
   a. Each IS is a processor with a bus to all cells. The IS processors are interconnected (e.g., by a bus, network, or shared memory). Each IS has a copy of the program being executed and can broadcast an instruction to all cells in unit time. The parallel execution of a command is SIMD in nature.
   b. Each cell listens to only one IS. Initially, all cells listen to the same IS. The cells can switch to another IS in response to commands from the current IS.
   c. The number of cells is much larger than the number of IS.
   d. An active cell executes the commands it receives from its IS while an inactive cell listens to but does not execute the commands from its IS. Each IS has the ability to unconditionally activate all cells listening to it.
3. ASSOCIATIVE PROPERTIES:
   a. An IS can instruct its active cells to perform an associative search. Successful cells are called responders while unsuccessful cells are called non-responders. The IS can activate either the set of responders or the set of non-responders. It can also restore the previous set of active cells. Each of these actions requires one unit of time.
   b. Each IS has the ability to select an arbitrary responder from the set of active cells in unit time.
   c. Each IS can instruct the selected cell to broadcast data on the bus. All other cells listening to this IS receive the value placed on the bus in unit time.
4. CONSTANT TIME GLOBAL OPERATIONS:
   a. An IS can compute the OR or AND of a value in all active PEs in unit time.
   b. An IS can compute the maximum or minimum of a value in a common record in each of its active PEs in a constant time.
5. CONTROL PARALLELISM:
   a. Cells without further work to do are called idle cells and are assigned to a specified IS which (among other tasks) manages the idle cells. An idle cell can be dynamically allocated to an IS in unit time. Any subset of cells can be deallocated and reassigned as idle cells in constant time.
   b. If an IS is executing a task that requires two or more subtasks involving data in disjoint subsets of the active cells, control (MIMD) parallelism can be invoked by assigning a subtask to an idle IS. When all of the subtasks generated by the original IS are completed, the cells are returned to the originating IS.

Figure 1 - The ASC Model

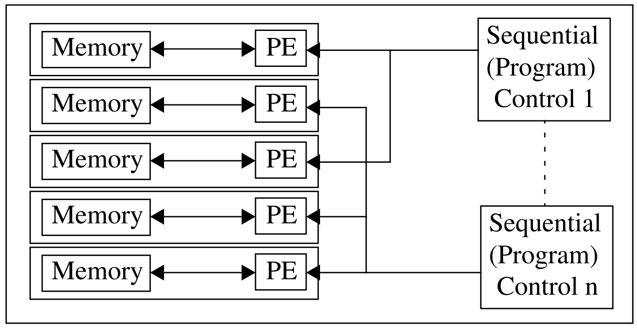


Figure 2 - Cellular Memory

ASSOCIATIVE PROGRAMMING TECHNIQUES

Frequently a few basic techniques determine the "feel" of a programming paradigm such as pointers in C, tail recursion or list processing in Lisp and Prolog. In ASC, the associative search is the fundamental operation and its influence is felt in constant time operations, tabular representation of abstract data structures, responder processing, and control parallelism. The simplest ASC model assumes only one instruction stream (IS). This model can be supported on existing SIMD computers, and is assumed in the remainder of this paper unless explicitly stated otherwise. Each of these areas are discussed below.

Constant Time Operation

Data parallelism is a basic model used in many languages. ASC uses data parallelism as the basis for associative searching which takes time proportional to the number of bits in a field, not the number of data items being searched. Thus, assuming that all the data fits in the computer, it executes in constant time[10] just as comparison, addition and other data parallel arithmetic operations do. In addition to basic searching for a pattern, the constant time functions[5], maximum, minimum, greatest lower bound and least upper bound are extensively used in ASC. The constant time functions have corresponding constant time associative cell index functions, (maxdex, mindex, prvdex, and nxtdex) which are used for associative reduction. For example, the query “What is the salary of the oldest employee?” requires a maximum search on the age field, but the associated salary, not the age, is the desired item. The maxdex function in “*salary*[maxdex(*age*$)]” expresses the association between the maximum age and the associated salary. The mindex, prvdex and nxtdex functions perform similarly. Computers with the properties specified in Figure 1 can execute these functions in constant time. In addition, today's sequential computers are powerful enough to effectively support these operations for many applications.

Tabular Data Structures and Structure Codes

Tabular data structures (i.e. tables, charts, and arrays) have two advantages for ASC. First, they are ubiquitous; they need only a minimal introduction to be effectively used. Second, the concept of processing an entire column of a table simultaneously is easy to comprehend. Tables and arrays are a common and natural organization for databases and many scientific applications.

There are a number of common abstract data structures including stacks, queues, trees and graphs which normally are implemented using address manipulation via pointers and indexes. Physical address relationships between data are not present in an associative processor. Instead structure codes, which are numeric representations of the abstract structural information, are associated with the data. The codes are generated automatically and appropriately named functions (e.g. parent(), sibling(), and child()) are used to manipulate them. The programmer need only be aware of the data structure type being used (e.g. tree, graph, etc.), and not the internal structure codes themselves.

One of the major advantages of structure codes is that they allow the data to be expressed in tabular form so that they can be processed in a data parallel manner. That means lists, trees, and graphs can be searched associatively in constant time instead of having to be sequentially searched element by element.

The simplest structure codes are those for arrays. For non-numerical applications, a vector can be represented by a row field and a value field as shown in Figure 3a. Likewise, a two dimensional array or matrix can be represented by a row field, a column field and a value field. The matrix value at position (1, 2) can be found to be 89 in Figure 3b in constant time by searching for row 1 and column 2 and retrieving the associated value.

row	value
1	95
2	17
3	36
4	47

3a) Vector

row	column	value
1	1	92
1	2	89
2	1	63
2	2	52

3b) Row-Column

Figure 3 - Structure Code

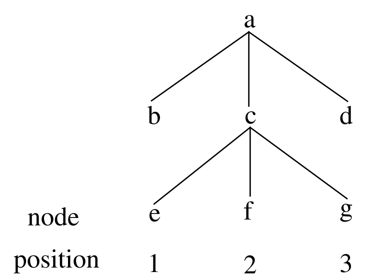
time	value
6	35
2	15
4	20
3	10
1	100

FIFO = MINIMUM(time);
LIFO = MAXIMUM(time);

Figure 4 - Associative Queues

Frequently, directly useful information can be used as the structure code. For example in Figure 4, the time of arrival is used to implement FIFO and LIFO queues. The instructions shown in Figure 4 use the maximum and minimum functions to retrieve their associated values.

More sophisticated structure codes are required for trees and graphs. If trees are put into a canonical form, and the position of the nodes on every level are numbered from left to right, a code for every node in the tree can be generated by starting at the root of the tree and listing the node numbers along the path to the node in question. If the code is left justified with zero fill, it will support parallel searching, concatenation, insertion and deletion. Figure 5 gives an example of a tree and its structure codes as represented in an associative memory. In Figure 5, the left and right siblings of node f may be found in constant time by using the sibdex function, sibdex(code[node$==‘f’]). This kind of operation is very useful for expression parsing[13].



code	node
1000	a
1100	b
1200	c
1300	d
1210	e
1220	f
1230	g

Figure 5 - Tree Structure Codes

Quadsected square codes are structure codes for graphs which can be used for the generation of node domination, node influencing and similar information useful in control flow graph analysis. A quadsected square is a square that has been divided into four subsquares. The quadrants of a quadsected square can be recursively subdivided to any level. The quadsected square code calculation and manipulation functions are performed in data parallel mode for all nodes of a graph. For example, given the code for a node, the dominance relationship between the node and all other nodes in the graph can be computed in constant time independent of the size of the graph.

Figures 6a and 6b illustrate the dual relationship between binary graphs and quadsected squares. The graph's binary fan-out (node 1 branches to 2 and 3) and binary fan-in (node 2 and 3 converge on 4) shown in Figure 6a are mapped onto the location code map shown in Figure 6b. A more complex example is given in Figures 6c, 6d and 6e where it can be seen that control flow starts in quadrant (A) the upper left most subdivision. It then recursively flows into the upper left most subdivision of the two adjacent quadrants (B of BCDE) and (F). Each subdivision continues this recursive process until the final two quadrants within a subdivision are joined at their right most subdivision (C and D are joined at E) and (E and F are joined at G).

The structure code, Figure 6d, for a recursively quadsected square is obtained by specifying the position of the top level subdivision first (as the left most digit) and then the position of the next recursive subdivision, and so on, with zero fill used on the right.

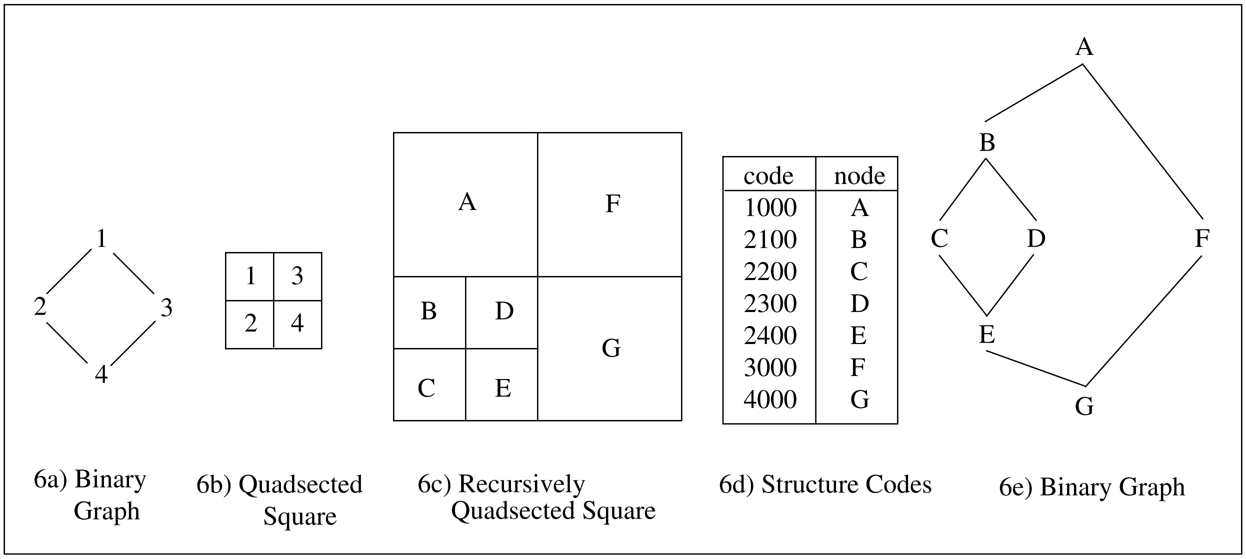


Figure 6 - Quadsected Square Encoding

Responder Processing

The result of an associative search is a “set of responders”. These responders are those cells that successfully matched the associative search query. Data parallel operations applied to a set of responders essentially act as substitutes for the index-based loops used in Fortran and C. However, it is sometimes desirable to process each responder individually. In responder iteration, a responder is chosen randomly from the set of responders. The information in the selected responder is then processed. The responder is released when it has been completely processed and the “next” responder is selected for processing. Responder iteration is an effective way of using parallel searching to avoid sorting unordered data. (The “loop while” statement in Figure 8 is an example of responder iteration.)

Responder processing can be used to achieve constant time memory allocation. Idle cells are assigned to a single IS. When one or more new cells are needed, they are selected at random from the idle pool and allocated to the requesting IS. When the cells are no longer needed, they are identified by associative search and released in parallel and returned to the idle IS. Associative memory allocation is efficient since allocating a cell is equivalent to allocating an entire record in conventional languages. Figure 7 illustrates the difference between associative memory allocation and C-based data parallel memory allocation where additional fields for the active processor, not cells, are allocated.

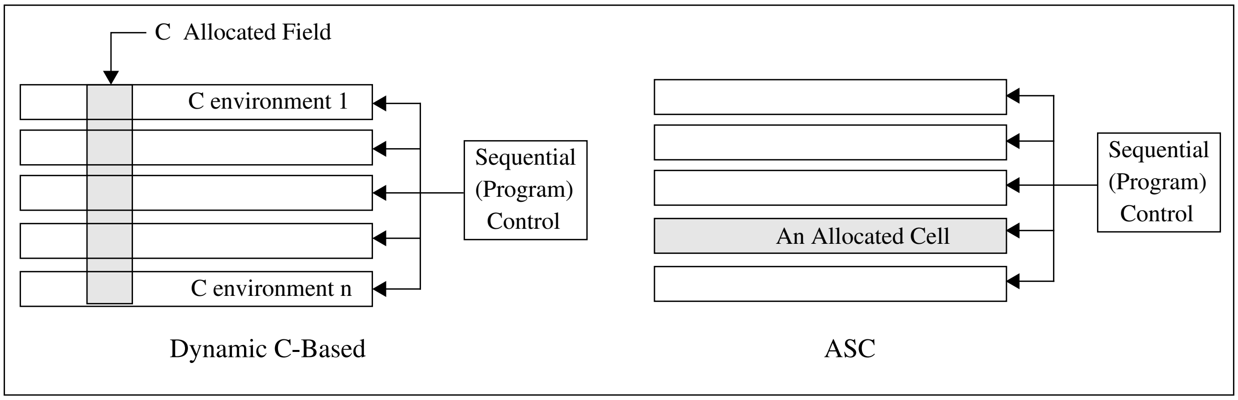


Figure 7 - Dynamic Memory Allocation

Control Parallelism

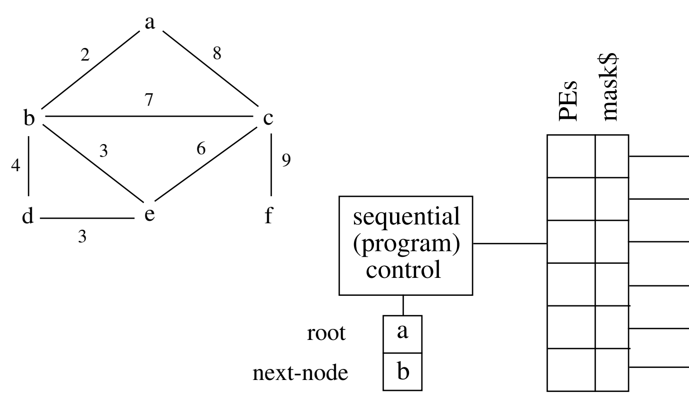
To this point the discussion has centered around data parallelism. However, the ASC model described in Figure 1 accommodates both data and control parallelism so that all of the computer’s cells can be efficiently utilized. The control parallel component is dependent on the dynamic manipulation of instruction streams (IS) in response to associative searches. The mechanism relies on partitioning the responders into mutually exclusive subsets. For example, the evaluation of an IF's conditional expression divides the active cells into two mutually exclusive partitions: One containing the cells which respond TRUE and one containing the cells which respond FALSE. These partitions can be processed using control parallelism by forking the process and assigning one IS to execute the THEN portion of the IF statement with the TRUE responders and another IS to execute the ELSE portion with the FALSE responders. The ISs execute in parallel and each in a data parallel mode. The programmer does not need any control parallel statements such as FORK or JOIN as the control parallelism is inherent in the statements. Case statements are another example of control parallelism, except instead of using two partitions as in the IF-THEN-ELSE there are *n* partitions - one for each of the *n* cases.

A significant speedup of up to *k* in the running time of certain algorithms is possible using an associative computer with a constant number *k* of instruction streams. Moreover, if the number of instruction streams is not restricted to being constant, then new algorithms with lower complexity times may be possible.

APPLICATIONS

The ASC paradigm has been applied to a wide range of applications, including image processing (e.g. convolution[10]), graph algorithms (e.g. the minimal spanning tree[10]), rule based inference engines (e.g. OPS5[6]), Associative Prolog[10], graphics (ray tracing[7]), data base management[8], compilation (first pass[1] and optimization[9]) and heterogeneous networks[11,12].

An ASC version of Prim's minimal spanning tree (MST) algorithm[14] using associative computing techniques with only one IS is given in Figure 8. The values given in Figure 8 indicate the state of the algorithm at the end of the first iteration through the "loop while". All of the statements in the algorithm execute in constant time. The data for each node is stored in a record and the records are stored with at most one record per cell. The cell variables are identified with a "$" symbol following the variable name. The cost of an edge from node x to node y is stored in the $x field of node y and the $y field of node x. Each record also has the additional variables, candidate$, parent$, and current_best$. Root and next-node are scalar variables. If root=n, then the terminology "root's field" refers to the field n$. Since one tree edge is selected by each pass through the loop and a spanning tree has *n-1* tree edges, the running time of this algorithm is *O(n)*. This is a cost optimal parallel implementation of Prim's original MST algorithm since it has a sequential running time of $O(n^2)$. There is no additional overhead incurred as the size of the graph increases, only additional cells are needed, so the algorithm is easily scalable to larger data sets. Moreover, since no networks are used and no task forking or join operations are needed, the communications and synchronization overhead costs are essentially eliminated.



node$	a$	b$	c$	d$	e$	f$	candidate$	parent$	current_best$
a	∞	2	8	∞	∞	∞	no		
b	2	∞	7	4	3	∞	no	a	2
c	8	7	∞	∞	6	9	yes	b	7
d	∞	4	∞	∞	3	∞	yes	b	4
e	∞	3	6	3	∞	∞	yes	b	3
f	∞	∞	9	∞	∞	∞	waiting		

```
ASC-MST-PRIM (root)
initialize candidates to "waiting"
if there are any finite values in root's field, then
    set candidate$ to "yes"
    set parent$ to root
    set current_best$ to the values in root's field
    set root's candidate field to "no"
loop while some candidate$ contain "yes"
    for them
        restrict mask$ to mindex(current_best$)
        set next_node to a node identified in the preceding step
        set candidate$ to "no"
        if the values in next_node's field are less than current_best$, then
            set current_best$ to value in next_node's field
            set parent$ to next_node
    if candidate$ is "waiting" and the value in next_node's field is finite
        set candidate$ to "yes"
        set parent$ to next_node
        set current_best$ to next_node
        set current_best$ to the values in next_node's field
```

Figure 8 - An Associative Minimal Spanning Tree Program

ASC has been combined with logic programming to achieve high performance intelligent reasoning, data parallel scientific computing, and efficient information retrieval from large knowledge bases[3]. The strategy in the design of the associative logic programming system is to maximize the use of bit-vector and data parallel operations, and minimize the movement of scalar data. Facts, relations, and the left hand sides of rules are represented as records (associations) of parallel fields with one record per cell. The right hand sides of the rules are compiled into an abstract instruction set. A simplified schematic of the model is given in Figure 9.

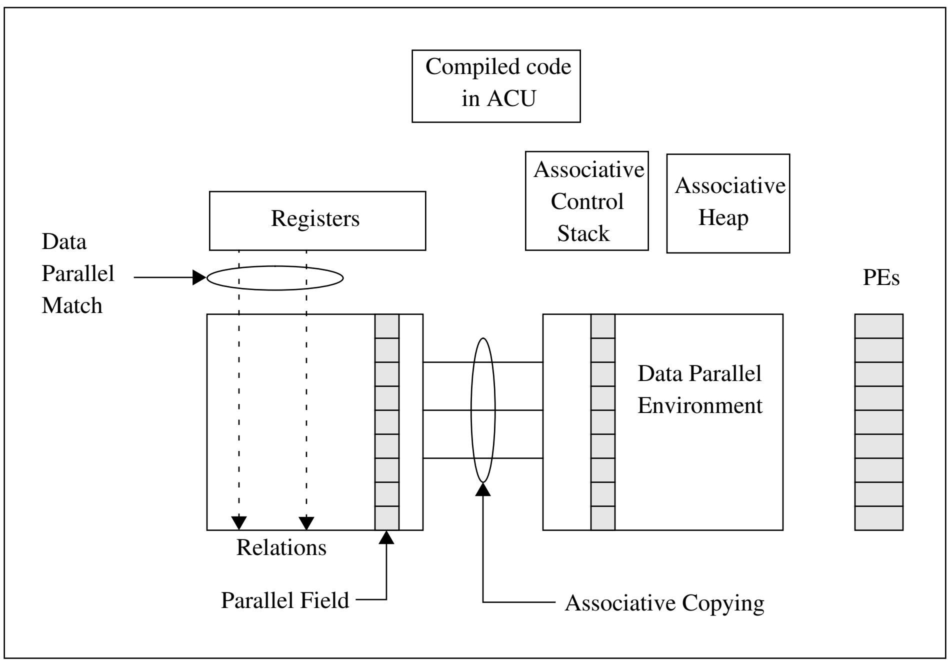


Figure 9 - Associative Logic Programming

Some of the advantages of combining associative and logic computing are: 1) the speed of knowledge retrieval is independent of the number of ground facts, 2) knowledge retrieval is possible even if the information is incomplete, making knowledge discovery possible, 3) relations with a large number of arguments are handled efficiently with little overhead, 4) associative lookup is fast, allowing the tight integration of high performance knowledge retrieval and data parallel computation without any overhead due to data movement or data transformation, and 5) the model is efficient for both scalar and data parallel computations on various abstract data types such as sequences, matrices, bags and sets.

The implication of these results is that the model can be successfully applied to data intensive problems such as geographical information systems, image understanding systems, statistical knowledge bases, and genome sequencing. For example, in geographical information systems, spatial data structures such as quad-trees and oct-trees are represented associatively with structure codes. As a result, different regions having the same values can be identified using associative search in constant time.

The integration of data parallel scientific computing, knowledge base retrieval, and rule based reasoning provides necessary tools for image understanding systems. Statistical queries can directly benefit from associative searches, associative representation of structures, data parallel arithmetic computations, and data parallel aggregate functions. Genome sequencing requires integration of knowledge retrieval, efficient insertion and deletion of data elements, and efficient manipulation of matrices for the heuristic matching of sequences.

IMPLEMENTATION

One intention of ASC is that it can be efficiently supported in hardware by a continuum of compute engines. The first step in this continuum has been to install the ASC language on conventional sequential computers (e.g. PCs and workstations). If needed, associative functions and operations can be sped up by using accelerator cards similar to the one currently being developed for ARPA[4] by Adaptive Solutions Inc. Conventional SIMD computers provide the third level of associative functionality. (ASC has been installed on the STARAN, ASPRO, Wavetracer, and the CM2.) The highest, most complex, and fastest level would be a multiple instruction stream SIMD computer based on the specifications in Figure 1.

Associative techniques are being used in distributed heterogeneous networks. A new programming paradigm called Heterogeneous Associative Computing[11] (HAsC) is presently under development at KSU. This paradigm takes from the ASC model the concept of cells and instruction broadcasting. It uses tabular data and massively parallel searches to match commands and data to machines. The result is that, in an extension of polymorphism, commands are executed on the machines best suited for them.

SUMMARY

The associative techniques of the 1970's augmented with new techniques such as structure codes, dynamic memory allocation, responder iteration, multiple instruction streams, associative selection and reduction notation and pronouns, form the basis of a programming paradigm which makes use of today's inexpensive computing power to facilitate parallel programming. The ASC paradigm uses a tabular data organization, massive parallel searching and simple syntax so that the paradigm is easily comprehensible to computer specialists and non-specialists alike. Furthermore, the ASC paradigm is suitable for all levels of computing from PCs and workstations to multiple instruction stream SIMDs and heterogeneous networks.

ACKNOWLEDGMENTS

The authors wish to thank Selim Akl for his helpful comments.

REFERENCES

1. Asthagiri, C. [1986], Context Sensitive Parsing Using an Associative Processor, M.S. Thesis, Kent State University, 1986.

2. Batcher, K. [1976], “STARAN Parallel Processor System Hardware,” Proc. NCC, 43, pp. 405-410.

3. Bansal, A., J. Potter and L. Prasad, 1992, “Data Parallel Compilation and Extending Query

Power of Large Knowledge Bases," in The Proceedings of the International Conference of Tools for Artificial Intelligence, pp. 276-283.

4. Electronic Engineering Times, Feb, 7, 1994, p. 41.

5. Falkoff, A. [1962], "Algorithms for Parallel_Search Memories," J. Associative Computing. Mach. 9, pp. 488-511.

6. Baker, J. W. and A. Miller, "A Parallel Production System Extending OPS5," *Proceedings on the Frontiers of Massively Parallel Computation*, J. JaJa, ed., 1990, pp. 110-118.

7. Krochta, T., Parallel Ray Tracing, Masters Thesis, Kent State University, Kent, Ohio, 1986.

8. Mamoozadeh, K., Relational Databases on Associative Processors, Masters Thesis, Kent State University, Kent, Ohio, 1986.

9. Miles, R., Optimizing Associative Intermediate Code, Masters Thesis, Kent State University, Kent, Ohio, 1993.

10. Potter, J. [1992], Associative Computing - A Programming Paradigm for Massively Parallel Computers, Plenum Publishing, New York, N.Y.

11. Scott, S.L., Potter, J., "Heterogeneous Associative Computing - HAsC", 2nd Associative Processing and Applications Workshop, Syracuse University, Syracuse, N.Y., July 21- 23, 1993.

12. Leangsuksun, C., and Potter J. L., Data Placement and Distribution (DPD) Analysis for Heterogeneous Computing Environment, Work in progress, Department of Mathematics and Computer Science, Kent State University, Kent, Ohio.

13. Asthagiri, C. and J. L. Potter, "Associative Parallel Common Sub-expression Elimination", Technical Report CS-9405-06, Department of Mathematics and Computer Science, Kent State University, Kent, Ohio.

14. Baker, J. W. and J. Potter, "A model of Computation for Associative Computing", Technical Report CS-9409-07, Department of Mathematics and Computer Science, Kent State University, Kent, Ohio, September 1994.