ASC: An Associative-Computing Paradigm

Jerry Potter, Johnnie Baker, Stephen Scott, Arvind Bansal,
Chokchai Leangsuksun, and Chandra Astagiri

Kent State University

Associative computing evolved in an era when associative memories were both relatively new and, because they required a comparator at each bit of memory, relatively expensive. In the early 1970s, Goodyear Aerospace improved upon early associative processing techniques with its Staran SIMD (single instruction, multiple data) computer. Goodyear realized that the massively parallel search capability of bit-serial SIMDs could simulate associative searching, with the cost advantage of sharing the comparison logic (that is, the processing elements) over all the bits in an entire row of memory. This approach provided two additional benefits. The word widths could be very large (from 256 bits to 64 kilobits), and the data could be processed in situ using the same PEs.

However, today’s lower hardware costs and increased computing speeds allow parallel techniques to be effectively emulated on conventional sequential machines. Accessing data by associative searching rather than addresses and processing data in memory require a new programming style. One goal of our research is to develop a parallel programming paradigm that is suitable for many diverse applications, is efficient to write and execute, and can be used on a wide range of computing engines, from PCs and workstations to massively parallel supercomputers.

Our associative-computing (ASC) paradigm is an extension of the general associative processing techniques developed by Goodyear. We use two-dimensional tables as the basic data structure. Our paradigm has an efficient associative-based, dynamic memory-allocation mechanism that does not use pointers. It incorporates data parallelism at the base level, so that programmers do not have to specify low-level sequential tasks such as sorting, looping, and parallelization.

Our paradigm supports all of the standard data-parallel and massively parallel computing algorithms. It combines numerical computation (such as convolution, matrix multiplication, and graphics) with nonnumerical computing (such as compilation, graph algorithms, rule-based systems, and language interpreters). This article focuses on the nonnumerical aspects of ASC.

The ASC model

The ASC model is the basis of a high-level associative-programming paradigm and language. As described in the sidebar, “Properties of the ASC model,” the extended model provides a basis for algorithm development and analysis similar to the
PRAM (parallel random-access memory) models, with the additional provisions that hardware can be built to support this model and that its primitive operations are sufficiently rich to allow efficient use of massive parallelism.

These features let us develop parallel algorithms for large problems that can be abstractly analyzed and executed. Furthermore, algorithms based on a common model will have greater applicability and retain their importance longer than ones 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. The simplest ASC model assumes only one instruction stream (IS). This model can be supported on existing SIMD computers and is assumed throughout unless we state otherwise.

The sidebar lists the specific properties that the hardware must have to support the model. Reflecting these specifications, the ASC language is characterized by built-in associative reduction notation, associative responder iteration, responder-based flow of control, 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 nonnumerical focus of ASC are the unique structure code features and dynamic memory allocation. The most important features of ASC are discussed below. (ASC language syntax is described in detail in Potter.)

**Properties of the ASC model**

We have applied the ASC paradigm to a wide range of applications, including image processing (for example, convolution), graph algorithms (for example, the minimal spanning tree), rule-based inference engines (for example, OPS81), Associative Prolog, graphics (ray tracing), database management, compilation (first pass and optimization), and heterogeneous networks.

Our intention is that ASC 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 such as PCs and workstations. Second, associative functions and operations can be sped up by using accelerator cards similar to the one currently being developed for ARPA by Adaptive Solutions Inc. Conventional SIMD computers provide the third level of associative functionality. (ASC has been installed on Staran, Aspro, WaveTracer, and the CM-2.) The highest, most complex, and fastest level would be a multiple instruction stream SIMD computer built to meet the specifications of the following computation model.

**Cell properties**

- Cells consist of a processing element (PE) and a local memory (see Figure A).
- The memory of an associative computer consists of an array of cells.
- There is no shared memory between cells. Each PE can only access the memory in its own cell. The cell’s PE can be interconnected with a modest network (for example, a grid).
- Related data items are grouped together (associated) into records and typically stored one per cell. We assume that there are more cells than data.

**Instruction stream (IS) properties**

- Each IS is a processor with a bus to all cells. The IS processors are interconnected (for example, 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.
- Each cell listens only to 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.
- The number of cells is much larger than the number of IS’s.
- 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.

**Associative properties**

- An IS can instruct its active cells to perform an associative search. Successful cells are called responders, while unsuccessful cells are called nonresponders. The IS can activate either the set of responders or the set of nonre-
that all the data fits in the computer, it executes in constant time, just as comparison, addition, and other data-parallel arithmetic operations do. In addition to basic pattern searching, ASC makes extensive use of constant time functions (maximum, minimum, greatest lower bound, and least upper bound). The constant time functions have corresponding constant-time associative index functions (maxdex, mindex, prdex, and ntxdex), 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. Computers with the properties specified in the sidebar can execute these functions in constant time. In addition, today’s sequential computers are powerful enough to emulate these operations for many problems.

Tabular data structures and structure codes. Tabular data structures (that is, tables, charts, and arrays) have two advantages for ASC. First, they are ubiquitous, and tables and arrays are a common and natural organization for databases and many scientific applications, and users need only a minimal introduction to manipulate them effectively. Second, the concept of processing an entire column of a table simultaneously is easy to comprehend.

There are a number of common abstract data structures, including stacks, queues, trees, and graphs, that are normally implemented using address manipulation via pointers and indexes. In an associative computer, in contrast, physical address relationships between data are not present. 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 — for example, parent(), sibling(), and child() — are used to manipulate them. The programmer need be aware only of the data structure type being used (tree, graph, and so forth) 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. This means that lists, trees, and graphs can be searched associatively in constant time instead of having to be sequentially searched element by element. Tabular organizations are stored one row per cell in an associative computer. Thus, any one field (a column of the table) can be searched in parallel by broadcasting the desired value to all cell PEs, which then compare it with their local values.

sponders. It can also restore the previous set of active cells. Each of these actions requires one unit of time.
• Each IS has the ability to select an arbitrary responder from the set of active cells in unit time.
• 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.

Constant time global operations
• An IS can compute the or or and of a binary value in all active PEs in unit time.
• An IS can identify the cells with the maximum or minimum value in each of its active PEs in constant time.

Control parallelism
• 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.
• 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 subtasks generated by the original IS are completed, the cells are returned to the originating IS.

A new programming paradigm called Heterogeneous Associative Computing (HASC) is presently under development at Kent State University. From the ASC model, this paradigm takes 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.

References

Additional information is available via WWW at http://nimitz.mcs.kent.edu/∼potter, ∼chokchai, ∼s-scott, ∼arvind, ∼baker.
The simplest structure codes are those for arrays. For nonnumerical applications, a vector can be represented by a row field and a value field as shown in Figure 1a. Likewise, we can represent a two-dimensional array or matrix by a row field, a column field, and a value field. In Figure 1b, the matrix value at position (1, 2) can be found in constant time by searching for row 1 and column 2 and retrieving the associated value — 89.

Frequently, we can represent directly useful information in the structure code. For example, the time of arrival is used to implement FIFO and LIFO queues. For example, in Figure 2 the FIFO value in the queue is retrieved using the mindex function to select the first (smallest or oldest) time entry and its associated value.

Trees and graphs require more sophisticated structure codes. If trees are put into a canonical form, and the position of the nodes on every level are numbered from left to right, we can generate a code for every node in the tree 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 3 gives an example of a tree and its structure codes as represented in an associative memory. The left and right siblings of node $f$ can be found in constant time by using the sibdex function — $\text{sibdex}(\text{code}[nose]=f)$.

This expression can be read from the inside out. First, the node field is searched for the value $f$; the response is used to select the associated structure-code value (1220), which is passed to the sibdex function. Sibdex combines the greatest-lower-bound and least-upper-bound search functions to identify codes 1210 and 1230 as being adjacent to 1220, and their associated nodes — e and g — as siblings of f.

All operations are constant time. This kind of operation is very useful for expression parsing.4

Quadsected square codes are structure codes for graphs that can be applied to the generation of node domination, node influencing, and similar information useful in control flow graph analysis. A quadsected square is a square 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 4a and 4b 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 4a are mapped onto the location code map shown in Figure 4b. A more complex example is given in Figures 4c, 4d, and 4e, where the control flow starts in quadrant A and flows into the upper left-most subdivision of the two adjacent quadrants (B of BCDE and D). 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).

We obtain the structure code for a recursively quadssected square (Figure 4d) by specifying the position of the top-level subdivision first (as the left-most digit), then the position of the next recursive subdivision, and so on, with zero fill used on the right.

Responder processing. The responders of an associative search are those cells that successfully matched the associative search query. Data-parallel operations
applied to the 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 arbitrarily selected and processed using both sequential and parallel operations. When processing is complete, the responder is idled and another responder is selected for processing. Responder iteration is an effective way of using parallel searching to avoid sorting unordered data.

We use responder selection to achieve constant-time memory allocation. Idle cells are assigned to a single instruction stream. When an IS needs one or more new cells, they are arbitrarily selected from the idle pool and allocated to the requesting IS. When that IS no longer needs those cells, they are identified by associative search, released in parallel, and returned to the idle IS. Figure 5 illustrates the difference between associative-memory allocation and C-based data-parallel memory allocation, where additional fields, not cells, are allocated to the active processors. (The “loop while” statement in Figure 6 is an example of responder iteration.)

**Control parallelism.** To this point our discussion has centered on data paral-

---

**Figure 6. An associative minimal spanning tree algorithm.**

```plaintext
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 candidates contain "yes"
  for them
    restrict mask$ to minindex(current_best$)
  set next_node to a node identified in the preceding step
set its candidate to "no"
if the values in next_node's field are less than current_best$, 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 values in next_node's field
```

---
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. The strategy in the design of the associative-logic programming system is to maximize the use of bit-vector and data-parallel operations and to 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 7.

Some 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.

These advantages suggest that this paradigm 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 quadrants and octtrees are represented associatively with structure codes. As a result, different regions having the same values can be identified using associative searches 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 knowl-
edge retrieval, efficient insertion and deletion of data elements, and efficient manipulation of matrices for the heuristic matching of sequences.

In the associative techniques of the 1970s 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 that 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 PC's and workstations to multiple instruction stream SIMD's and heterogeneous networks. 

Acknowledgments

The authors thank Selim Akil for his helpful comments. Stephen L. Scott's work on heterogeneous associative computing is supported by a NASA GSRP-HFCC fellowship and by NraD.

References


November 1994