Chapter 16

The User-Level Virtual Machine

The virtual machine provided by the hardware and operating system to a single user task consists of a control processor acting as a partition manager (PM), a set of processing nodes, and facilities for interprocessor communication. Each node is an ordinary general-purpose microprocessor capable of executing code written in C, Fortran, or assembly language. The processing nodes may also have optional vector units for high arithmetic performance.

The operating system is CMOST, a version of SunOS enhanced to manage CM-5 processor, I/O, and network resources. The PM provides full UNIX services through standard UNIX system calls. Each processing node provides a limited set of UNIX services.

A user task consists of a standard UNIX process running on the PM and a process running on each of the processing nodes. Under timesharing, all processors are scheduled *en masse*, so that all are processing the same user task at the same time. Each process of the user task, whether on the PM or on a processing node, may execute completely independently of the rest during their common time slice.

The Control Network and Data Network allow the various processes to synchronize and transfer data among themselves. The unprivileged control registers of the Network Interface hardware are mapped into the memory space of each user process, so that user programs on the various processors may communicate without incurring any operating system overhead.

Copyright © 1993 Thinking Machines Corporation

16.1 Communications Facilities

Each process of a user task can read and write messages directly to the Control Network and the Data Network. The network used depends on the task to be performed.

The Control Network (CN) is responsible for communications patterns in which many processors may be involved in the processing of each datum. One example is broadcasting, where one processor provides a value and all other processors receive a copy. Another is reduction, where every processor provides a value and all values are combined to produce a single result. Values may be combined by summing them, finding the maximum input value, or taking the logical OR or exclusive OR of all input values; the combined result may be delivered to a single processor or to all processors. (Software provides minimum-value and logical AND operations by inverting the inputs, applying the hardware maximum-value or logical OR operation, then inverting the result.) Note that the control processor does not play a privileged role in these operations; a value may be broadcast from, or received by, the control processor or any processing node with equal facility.

The Control Network contains integer and logical arithmetic hardware for carrying out reduction operations. This hardware is distinct from the arithmetic hardware of the processing nodes; CN operations may be overlapped with arithmetic processing by the processors themselves. The arithmetic hardware of the Control Network can also compute various forms of parallel prefix operations, where every processor provides a value and receives a result; the *n*th result is produced by combining the first *n* input values. Segmented parallel prefix operations are also supported in hardware.

The Control Network provides a form of two-phase barrier synchronization (also known as “fuzzy” or “soft” barriers). A processor can indicate to the Control Network that it is ready to enter the barrier. When all processors have checked in, the Control Network relays this fact to all processors. A processor can thus overlap unrelated processing with the possible waiting period between the time it has checked in and the time it has been determined that all processors have checked in. This allows thousands of processors to guarantee the ordering of certain of their operations without ever requiring that they all be exactly synchronized at one given instant.

The Data Network is responsible for reliable, deadlock-free point-to-point transmission of tens of thousands of messages at once. Neither the senders nor the receivers of messages need be globally synchronized. At any time, any processor may send a message to any processor in the user task. This is done by

November 1993
Copyright © 1993 Thinking Machines Corporation

writing first the destination processor number, and then the data to be sent, to control registers in the Network Interface (NI). Once the Data Network has accepted the message, it assumes all responsibility for eventual delivery of the message to its destination. In order for a message to be delivered, the processor to which it was sent must accept the message from the Data Network. However, processor resources are not required for forwarding messages. The operation of the Data Network is independent of the processing nodes, which may carry out unrelated computations while messages are in transit.

There is no separate interface for special patterns of point-to-point communication, such as nearest neighbors within a grid. The Data Network presents a uniform interface to the software. The hardware implementation, however, has been tuned to exploit the locality found in commonly used communication patterns.

There are two mechanisms for notifying a receiver that a message is available. The arrival of a message sets a status flag in a Network Interface control register; a user program can poll this flag to determine whether an incoming message is available. The arrival of a message can also optionally signal an interrupt. Interrupt handling is a privileged operation, but the operating system converts an arrived-message interrupt into a signal to the user process. Every message bears a four-bit tag; under operating system control, some tags cause message-arrival interrupts and others do not. (The operating system reserves certain of the tag numbers for its own use; the hardware signals an invalid-operation interrupt to the operating system if a user program attempts to use a reserved message tag.)

The Control Network and Data Network provide flow control autonomously. In addition, two mechanisms exist for notifying a sender that the network is temporarily clogged. Failure of the network to accept a message sets a status flag in a Network Interface control register; a user program can poll this flag to determine whether a retry is required. Failure to accept a message can also optionally signal an interrupt.

Data can also be transferred from one user task to another, or to and from I/O devices. Both kinds of transfer are managed by the operating system using a common mechanism. An intertask data transfer is simply an I/O transfer through a named UNIX pipe.

Copyright © 1993 Thinking Machines Corporation

16.2 Data Parallel Computations

While the user may code arbitrary programs for the various processors and put the general capabilities of the Network Interface to any desired use, the CM-5 architecture is designed to support especially well the data parallel model of programming. Parallel programs are often structured as alternating phases of local computation and global communication. Local computation consists of operations by each processor on the data in its own memory. Global communication includes any transfer of data between or among processors, possibly with arithmetic or logical computation on the data as it is transferred. By managing data transfers globally and coherently rather than piecemeal, the data parallel model often realizes economies of scale, reducing the overhead of synchronization for interprocessor communication. Frequently used patterns of communication are captured in carefully tuned compiler code generators and run-time library routines; they are presented as primitive operators or intrinsic functions in high-level languages so that the programmer need not constantly reinvent them.

The following sections discuss various aspects of the data parallel programming model and sketch the ways in which each is supported by the CM-5 architecture and communications structure.

Elemental and Conditional Computations

Elemental computations, which involve operating on corresponding elements of arrays, are purely local computations if the arrays are divided in the same way among the processors. If two such matrices are to be added together, for example, every pair of numbers to be added reside together in the memory of a single processing node, and that node takes responsibility for performing the addition.

Because each processing node executes its own instruction stream as well as processing its own local data, conditional operations are easily accommodated. For example, one processing node might contain an element on the boundary of an array while another might contain an interior element; certain filtering operations, while allowing all elements to be processed at once, require differing computations for boundary elements and interior elements. In the CM-5 data parallel architecture, some processors can take one branch of a conditional and others can take a different branch simultaneously with no loss of efficiency.

November 1993
Copyright © 1993 Thinking Machines Corporation

Replication

Replication consists of making copies of data. The most important special case is broadcasting, in which copies of a single item are sent to all processors. This is supported directly in hardware by the Control Network.

Another common case is spreading, in which copies of elements of a lower-dimensional array are used to fill out the additional dimensions of a high-dimensional array. For example, a column vector might be spread into a matrix, so that each element of the vector is copied to every element of the corresponding row of the matrix. This case is handled by a combination of hardware mechanisms.

If the processors are partitioned into clusters of differing size, such that the network addresses within each cluster are contiguous, then one or two parallel-prefix operations by the Control Network can copy a value from one processor in each cluster to all others in that cluster with particular speed.

Reduction

Reduction consists of combining many data elements to produce a smaller number of results. The most important special case is global reduction, in which every processor contributes a value and a single result is produced. The operations of integer summation, finding the integer maximum, logical OR, and logical exclusive OR are supported directly in hardware by the Control Network. Floating-point reduction operations are carried out by the nodes with the help of the Control Network and Data Network.

A common operation sequence is a global reduction immediately followed by a broadcast of the resulting value. The Control Network supports this combination as a single step, carrying it out in no more time than a simple reduction.

The cases of reduction along the axes of a multidimensional array correspond to the cases of spreading into a multidimensional array and have similar solutions. The rows of a matrix might be summed, for example, to form a column matrix. This case is handled by a combination of hardware mechanisms.

If the processors are partitioned into clusters of differing size, such that the network addresses within each cluster are contiguous, then one or two parallel-prefix operations by the Control Network can reduce values from all processors within each cluster and optionally redistribute the result for that cluster to all processors in that cluster.

Copyright © 1993 Thinking Machines Corporation

Permutation

The Data Network is specifically designed to handle all cases of permutation, where each input value contributes to one result and each result is simply a copy of one input value. The Data Network has a single, uniform hardware interface and a structure designed to provide especially good performance when the pattern of exchange exhibits reasonable locality. Both nearest-neighbor and nearest-but-one-neighbor communication within a grid are examples of patterns with good locality. These particular patterns also exhibit regularity, but regularity is not a requirement for good Data Network performance. The irregular polygonal tesselations of a surface or a volume that are typical of finite-element methods lead to communications patterns that are irregular but local. The Data Network performs as well for such patterns as for regular grids.

Parallel Prefix

Parallel prefix operations embody a very specific, complex yet regular, combination of replication and reduction operations. A parallel prefix operation produces as many results as there are inputs, but each input contributes to many results and each result is produced by combining multiple inputs. Specifically, the inputs and results are linearly ordered; suppose there are n of them. Then result j is the reduction of the first j inputs; it follows that input j contributes to the last $n–j+1$ results. (For a reverse parallel prefix operation — also called a parallel suffix operation — these are reversed: result j is the reduction of the last $n–j+1$ inputs, and input j contributes to the first j results.)

The Control Network handles parallel prefix (and parallel suffix) operations directly, in the same manner and at the same speed as reduction operations, for integer and logical combining operations. The input values and the results are linearly ordered by network address.

The Control Network also directly supports segmented parallel prefix operations. If the processors are partitioned into clusters of differing size, such that the network addresses within each cluster are contiguous, then a single Control Network operation can compute a separate parallel prefix or suffix within each cluster.

More complex cases of parallel prefix operations, such as on the rows or columns of a matrix or on linked lists, are variously handled through the Control Network or Data Network in cooperation with the nodes.

November 1993
Copyright © 1993 Thinking Machines Corporation

Virtual Processors

Data parallel programming provides the high-level programmer with the illusion of as many processors as necessary; one programs as if there were a processor for every data element to be processed. These are often described as *virtual processors*, by analogy with conventional virtual memory, which provides the illusion of having more main memory than is physically present.

The CM-5 architecture, rather than implementing virtual processors entirely in firmware, relies primarily on software technology to support virtual processors. CM-5 compilers for high-level data parallel languages generate control-loop code and run-time library calls to be executed by the processing nodes. This provides the same virtual-processor functionality made available by the Paris instruction set on the Connection Machine models CM-2 and CM-200, but adds further opportunities for compile-time optimization.

16.3 Low-Level User Programming

Low-level programs may be written for the CM-5 in C or Fortran 77. Assembly language is also available, though C should be adequate for most low-level purposes; all hardware facilities are directly accessible to the C programmer. A special assembler allows hand-coding of individual vector instructions for the processing nodes. Such instructions may be assembled separately or inserted directly into C code.

One writes low-level programs as two pieces of code: one piece is executed in the control processor, and the other is replicated at program start-up and executed by each processing node. One speaks of writing a program in "C & C" (a C program for the control processor and a C program for the nodes); one may also write in "Fortran & Fortran" ("F & F"), in "C & assembler," etc.

A package of macros and run-time functions supports common communications operations within a message-passing framework (see Chapter 14). Such low-level communications access allows the user to experiment with program organizations other than data parallel, to port programs easily from MIMD architectures, and to implement new primitives for use in high-level programs.

Copyright © 1993 Thinking Machines Corporation