The Space Congress® Proceedings

1987 (24th) Space - The Challenge, The Commitment

Apr 1st, 8:00 AM

The Use of Transputers in Processing Telemetry Data

Hugo M. Delgado

Electronic Engineer Engineering Development Directorate NASA, John F. Kennedy Space Center Florida

THE USE OF TRANSPUTERS IN PROCESSING TELEMETRY DATA

Hugo M. Delgado Jr.
Electronic Engineer
Engineering Development Directorate
NASA, John F. Kennedy Space Center Florida

ABSTRACT
Parallelism will be an essential ingredient of high performance systems of the future. The Inmos transputer is a high performance single-chip computer whose architecture facilitates the construction of parallel processing systems. Occam is a high level language developed for use with the Inmos transputer. This paper describes a project to evaluate the feasibility of using the transputer to implement real time processing of telemetry data.

INTRODUCTION

Since John Von Neuman discovered the principles over 40 years ago, all digital computers have been designed in a fundamentally similar way. A processor which can perform a set of basic numeric manipulations is connected to a memory system which can store numbers. Some of these numbers are data which the computer is required to process.
The other numbers are instructions to the processor and tell it which of the basic manipulations to perform. The instructions are passed to the processor one after the other and executed. Execution of a computer program is sequential, consisting of a series of primitive actions following one another in time.

The earliest digital computers were programmed using the basic numeric instructions understood by the processor. Such programming is so tedious and error prone that computer scientists soon began to design "high-level" languages, starting with Fortran and leading to the current proliferation which includes Basic, Pascal, Modula 2, C, Ada, Forth, Lisp, Prolog and hundreds of others.

These languages allow programmers to express the logic of a program in notations which use readable English (or French etc...) words, although with a tightly constrained and reduced syntax. A program called a compiler then translates these notations into the basic numeric instructions which the computer understands. For the majority of languages,

the product of the compiler is again a sequence of instructions, to be executed one at a time by the processor just as if they had been produced by hand. In other words, these languages faithfully reflect the nature of the underlying sequential Von Neuman computer in a form more palatable to human programmers.

To adequately model the concurrency of the real world, it would be preferable to have many processors all working at the same time on the same program. There are also huge potential performance benefits to be derived from such parallel processing. For regardless of how far electronic engineers can push the speed of an individual processor, ten of them working concurrently will still execute ten times as many instructions in a second.

Conventional programming languages are not well equipped to construct programs for such multiple processors, as their very design assumes the sequential execution of instructions. Some languages have been modified to allow concurrent programs to be written, but the burden of ensuring that concurrent parts of the program are synchronized (i.e. that they cooperate rather than fight) is placed on the programmer. This leads to such programming being perceived as very much more difficult than ordinary sequential programming.

The Inmos transputer, using the Occam language, is the first product to be based upon the concept of parallel, in addition to sequential, execution, and to provide automatic communication and synchronization between concurrent processes.

TRANSPUTER ARCHITECTURE

A transputer is a microcomputer with its own local memory and with links for connecting one transputer to another transputer. The transputer architecture defines a family of programmable VLSI components. The definition of the architecture falls naturally into the "logical" aspects which define how a system of interconnected transputers is designed and programmed, and the "physical" aspects which define how transputers, as VLSI components, are interconnected and controlled.

A typical member of the transputer product family is a single chip containing processor, memory, floating point processor (T800 only), and communications links which

provide point-to-point connection between transputers. In addition, each transputer product contains special circuitry and interfaces adapting it to a particular use. For example, a peripheral control transputer, such as a graphics or disk controller, has interfaces tailored to the requirements of a specific device. Presently there are three versions of the transputer:

T212 - 16 bit address lines
16 bit data lines
2K byte internal RAM
4 serial links
17.5MHz or 20MHz internal clock speed

T414 - 32 bit address lines
32 bit data lines
2K byte internal RAM
4 serial links
15MHz or 20MHz internal clock speed

T800 - 32 bit address lines
32 bit data lines
4K byte internal RAM
4 serial links
floating point processor
20MHz or 30MHz internal clock speed

On a chip, over 250,000 components provide all the resources of the processing engine, memory interface, and concurrent communications. The processor uses its 50-nanosecond cycle time to wring 10 million instructions a second out of its on-chip static RAM. For more memory-intensive programs, the processor brings to bear its 26-megabyte-per-second memory interface to access up to 4 gigabytes of off-chip memory.

One fundamental reason for the performance and implementation efficiency of the transputer is its approach to its instruction set, which resembles that of a reduced instruction set computer (RISC). Instead of a minicomputer-like instruction set with its multiple addressing modes, two or three address instruction, and extensive register set, the transputer has a lean, tightly encoded, and simple set of instructions. At 33 ns (T800-30MHz) per instruction, programs execute very quickly indeed, and the tight encoding significantly improves code density.

TRANSPUTER BUSES

The transputer uses three major buses to interconnect all the processor's registers. Two of these buses ship operands

to the arithmetic and logic unit, and the other carries the ALU result. A number of additional data paths allow certain registers to communicate directly, independent of transactions on the primary buses. These additional data paths also serve as routes for direct transfers between buses. The data-path registers are all specific to various functions of the transputer and include logic to perform certain data manipulations. The additional data paths allow multiple data transfers to occur in a single microcycle.

Unlike conventional microprocessors, the transputer includes no programmer-visible register for peripheral functions like communications. Programmers use Occam's INPUT and OUTPUT primitives to send messages through interprocess communication channels. The transputer specifies the state of an executing process with a simple six-register set, an instruction pointer, an operand and register, and a workspace pointer, which indicates the area of memory where the local variables of the process are stored. The last three registers belong to a small stack attached to the ALU. Instructions affecting this stack fall into two major classes, one-address and zero-address.

In one-address instructions, the top of the stack is one (implied) operand. The other is generally a location in the current workspace, although it can be a literal. Zero-address instructions use the stack for all operands and results. For example, the ADD instruction adds the top two values in the evaluation stack and places the result on top of it. The three-word stack removes any need for instructions to respecify an operand's address and provides a good balance between code density, process-switch time, and implementation complexity.

SILICON SCHEDULER

Still, to work within Occam's process-oriented design, the transputer incorporates a hardware process scheduler to multiplex its processor among a number of active processes, which may exist as normal or as priority processes. Priority processes are typically used for interrupt handling chores and may receive messages (interrupts) from another process, a communication link, the peripheral interface, the on-chip timer, or from the external event pin. If a normal process is being executed and a message arrives for a priority process, the processor starts to execute it within 6 ns typically. The execution of the normal process resumes only when no higher-priority process can proceed.

A process may be unable to proceed because it is waiting for I/O or the timer. When a process is blocked, its instruction pointer is saved in its workspace and restored when the process resumes. For each priority level, two registers maintain a list which chains together the workspaces of the processes that can proceed. One register points to the front of the list, the other to its end.

COMMUNICATION LINKS

The transputer's four links provide efficient point-to-point communication for intertransputer communications. Each link consists of two unidirectional signal lines that carry both data and control. The range of these TTL-compatible signals can be extended by the insertion of line drivers and receivers. Regardless of any transputer's internal speed all links will run at 10 megabits/sec or 20 megabits/sec, depending on the configuration of the link's speed pins.

Each transputer link supports memory-to-memory block transfer for on and off-chip memory. A link controller accepts a pointer and block count when an Occam INPUT or OUTPUT statement refers to an intertransputer channel. The direct memory access (DMA) transfer is totally asynchronous, with message transfer taking place totally independently of the processor. A memory arbitration unit coordinates and assigns priorities to access memory from the processor, links, and the peripheral interface. Operating simultaneously, all the links can transfer data concurrently with processor execution, for a peak throughput of more than 10 megabytes/sec.

Regardless of the word length of the communicating devices, a message is transmitted as a sequence of bytes through the link on a pair of wires. For transfer in a single direction, the sending transputer initiates traffic by transmitting a byte on one wire. The sender then waits for acknowledgment, which is sent through the other wire and which signifies that the receiving link can receive another byte and that a process is waiting to receive it. The sending link reschedules the sending process only after it has received an acknowledgment for the final byte of the message.

For duplex communication on a single link, a transputer interleaves the byte it is sending with acknowledgments for

the bytes it is receiving on the other link wire. An acknowledgment can be transmitted as soon as the reception of a data byte starts if there is room to buffer another one. Transmission can therefore be continuous, without any delay between data bytes.

OCCAM

In Occam, processes are connected to form concurrent systems. Each process can be regarded as a black box with internal state which can communicate with other processes using the point-to-point channels. Processes can be used to represent the behavior of many things; for example, a logic gate, a microprocessor, a machine tool or an office.

The processes themselves are finite. Each process starts, performs a number of actions, and then terminates. An action may be a set of sequential processes performed one after another, as in conventional programming language, or a set of parallel processes to be performed at the same time as one another. Since a process is itself composed of processes, some of which may be executed in parallel, a process may contain any amount of internal concurrence, and this may change with time as processes start and terminate.

The key concept is that communication is synchronized and unbuffered. If a channel is used for input in one process and output in another, communication takes place when both processes are ready. The value to be output is copied from the outputting process to the inputting process, and the inputting and outputting processes then proceed. Thus, communication between processes is like the handshake method of communication used in hardware systems. Since a process may have internal concurrency, it may have many input channels and output channels performing communications at the same time.

Occam can be used to program an individual transputer or to program a network of transputers. When Occam is used to program an individual transputer, the transputer shares its time between the concurrent processes and channel communication is implemented by moving data within the memory. When Occam is used to program a network of transputers, each transputer executes the process allocated to it. Communication between Occam processes on different transputers is implemented directly on transputer links. Thus, the same Occam program can be implemented on a variety of transputer configurations, with one optimized for cost,

another for performance, or another for an appropriate balance of cost and performance.

PRIMITIVE PROCESSES AND CONSTRUCTORS

Occam programs are constructed from three primitive processes: assignment, input and output. The assignment

```
v := e
```

sets the variable v to the value of the expression e. The output

```
c ! e
```

outputs the value of e to the channel c, and the input

```
c ? v
```

sets the variable v to a value input from the channel c.

Constructors are used to combine processes to form larger processes. The SEQuential constructor causes its components to be executed one after another, terminating when the last component terminates. The PARallel constructor causes its components to be executed concurrently, terminating only after all of the components have terminated. The ALTernative constructor chooses one component process for execution, terminating when the chosen component terminates. Finally, IF and WHILE constructs are provided.

The example of the sequential construct below is a simple buffer which repeatedly inputs a value from the channel buffer.in, then outputs it to the channel buffer.out. The sequential construct ensures that the input is complete before the output starts. WHILE TRUE causes the whole sequential construct to be executed repeatedly. VAR x : introduces the variable x for use in the input and output processes.

```
WHILE TRUE
  VAR x :
  SEQ
    buffer.in ? x
    buffer.out ! x
```

Occam has a simple, regular syntax. Each primitive process and each constructor occupies a line by itself, and the components of a construct are indented. Declarations, like VAR, are prefixed to constructs. The variables they declare have the scope of the construct to which the declaration is prefixed.

The parallel constructor is often used to combine sequential processes. In the example below it combines two simple

buffer processes to form a buffer which can hold two values. Each of the two simple buffer processes has its own variable x, and the buffers communicate using the channel comms. CHAN comms introduces the channel comms for use between the simple buffer processes.

```
CHAN comms :
PAR
  WHILE TRUE
    VAR x :
    SEQ
      buffer.in ? x
      comms ! x
  WHILE TRUE
    VAR x :
    SEQ
      comms ? x
      buffer.out ! x
```

TELEMETRY PROCESSING

One of the capabilities that ground stations like Kennedy Space Center (KSC) and Johnson Space Center (JSC) must have is the ability to process data transmitted from the flight hardware. Telemetry data received by the ground stations is in PCM (pulse code modulation) form. As an example, the Space Shuttle PCM stream major frames occur once a second. Within major frames there are minor frames which occur every 10ms (milli-seconds). The ground systems "lock-on" to the data stream when they detect and recognize major frame and minor frame sync. The actual data starts after each minor frame sync and is in consecutive bytes with 160 bytes per minor frame. The bytes can contain eight measurements per byte, as is the case of discrete type measurements, or can be one of the bytes of a multiple byte measurement. Eventually all 128Kbits/sec. of the PCM stream must be processed and presented to the user.

Traditionally, KSC processing of Orbiter PCM data has been done in the Firing Room. The Firing Room has three Front End Processors (FEP) to process the incoming PCM stream. For Space Station elements, the processing of the telemetry data will be done by the Ground Data Management System (GDMS). The subset of GDMS that will process PCM data will be the Data Acquisition Module (DAM).

The DAM provides the unique hardware interface between the GDMS and the incoming PCM stream. The functions that will be performed on the data include data acquisition, various data

checks, updating of values in the rest of GDMS, and initiation of notification messages when data values are out of limits.

There are four basic data checks and manipulations that must be performed on the incoming data. These are compression, limit checking, linearization, and trend checking.

Compression is the function of reducing the data rate by transmitting only those data values which have changed by a significant amount. There are tables which define the significant change value for each measurement.

Limit checking is the function of testing an incoming data value against pairs of high and low limits and notifying the responsible application program if it is out of those limits. These limits are contained in a table and are subject to real-time update by operator or program request. There are three set of limits: design limits (specified by the design agency), test limits (imposed by the test agency), and program limits (set by a user or a program).

Linearization is the function of converting incoming raw non-linear data values to a linear form. It also removes offset. This function requires calculating up to a fifth order polynomial with 32 bit real coefficients in real-time (I.E.
y = A5(x**5) + A4(x**4) + A3(x**3) + A2(x**2) + A1(x) + A0
Where A0 thru A5 are 32 bit real coefficients and "x" is the data received from the PCM stream.)

Trend checking is the function of estimating the value of the measurements at some delta time in the future and warning of potential violations.

It is expected that the Space Station telemetry data rates will be higher than present Orbiter data rates. There is a high possibility that the data rate will be 256Kbits/sec. or higher. Assuming that the data rate is 256Kbits/sec. Then the incoming bytes that must be processed arrive at a rate of 31.25 micro-seconds/byte. This implies that the compression, limit checking, trend checking, and linearization must be completed before the next byte arrives. It is obvious that parallel processing and high performance processing is needed. At the moment the Inmos transputer and Occam have proven the best product for this task.

Present development at KSC has shown that the fifth order polynomial calculation using the T414-20 (20 Mega-hertz) requires about 100 micro-seconds. The T414-20 uses software routines to perform the 32 bit multiplication. The T800-30 (30 Mega-hertz) will be available from Inmos in the fall of 1987. It is expected that the T800-30, with the hardware floating point processor, will take between 10-15 micro-seconds to perform the fifth order polynomial.

The present design of the Filter Card (hardware card that performs compression, limit checking, trend checking, and linearization) calls for six transputers working in parallel. The theoretical limit of the card for processing telemetry data is 1.6 Mega-bits/second. Further development and testing will determine the actual limit of the card.