Performance, pricing, and availability considerations make the Inmos Transputer a natural contender for robotic chores requiring parallel computation. One such motion-control application—the control of a juggling apparatus—uses Transputer-based processor networks based around Evergreen Design's XP/DCS (Transputer/Distributed Control System) engineered in conjunction with the Yale Robotics Laboratory. The Transputer is supported by an integrated software development system well-suited for parallel and concurrent versions of numerically complex operations. It is thus possible to experiment with motion-control algorithms without the need for writing real-time operating system kernels or developing parallel or concurrent code-distribution tools. Further, the Transputer comes with its own high-level language (Occam) that supports parallel processing as well as the TDS (Transputer Development System) for writing, distributing, and debugging code.

Robotic computational requirements are stringent. Typical servo motors have time constants of 10 to 100 msec, depend on load. And, while the Nyquist sampling rate is merely double the fastest frequency of the system to be controlled, a rate an order of magnitude higher is generally required. Hence, 1 kHz is a desirable update and sampling frequency. Further, a typical nonlinear control law for tracking reference trajectories with a six-degree-of-freedom industrial robot manipulator requires roughly $10^3$ floating-point operations every sampling interval. Such an algorithm would require 1 MFLOPS of raw computational capability for real-time implementation. Assuming that the controller must also sample joint positions and velocities and deliver torque commands to all six degrees of freedom, the algorithm must assume that 18 parallel I/O operations are performed at 1 kHz. Autonomous robots in environments cluttered with obstacles call for even greater power. Further, computational requirements in emerging smart-sensor applications will need distributed processing and multiple-channel I/O.

The Burden of Parallelism

The Transputer lets the designer place the burdens of parallelism (e.g., interprocessor communications, software development support) on a commercial product, while concentrating on customizing the computational "identity" of particular nodes with special-purpose hardware. With fast on-chip RAM interrupt, and DMA support, the 32-bit T800 Transputer (Figure 1) offers four DMA serial interprocessor communication links. These links circumvent constraints of bus-based interprocessor communication schemes with regard to reconfigurability and bandwidth. The result is a topology to which nodes can be added or deleted by connecting a four-wire serial cable (and system service connections).

Through the parallel processing constructs of the associated programming language, Occam, software requirements of process concurrency are readily addressed. Whether multitasking on one Transputer, or engaged in parallel implementation on a network of Transputers, the desired relationships between software processes and hardware processors can be easily specified. Meanwhile, a PC-based Transputer Development System,
The CPU board supports the T800 and T414 at jumper-selectable clock speeds of 15 to 20 MHz. An on-board signal ensures the processor is reset properly at power-up. Without this provision, random external bus accesses may occur until a program is downloaded. In motion-control applications, where I/O devices may be accessed arbitrarily, this could be disastrous.

Although the 128-Kbyte no-wait-state SRAM may seem inadequate for some applications, one of the benefits of employing distributed processing is the reduction in program size running on any given node. If data logging is desired, the 2-Mbyte BOO4 host memory can be used. The I/O board bus interface passes those signals required for additional memory, if needed.

The Transputer's four serial links are made available on the rear edge connector, which is compatible with the ITEM boards. Link speeds are user selected. To improve impedance matching in critical cases, the output lines' series resistances can be altered. Two of the links can be routed to fiber-optic ports on the board's front edge to ensure signal integrity in harsh emi settings. Current hardware limits fiber bandwidth to 5 Mb/s/sec and length to 17 meters.

A latched 32-bit bidirectional I/O bus provides support for a virtually unlimited number of I/O devices with minimal chip count. Six individually addressable sets of four latched handshaking output lines (plus eight handshaking input lines) further reduce the customizing effort needed for a specific I/O device. To prevent arbitrary latched outputs of the handshaking output lines at power-up (which could cause such disasters as enabling a robot joint while the torque command is not under control), these outputs "wake up" in a high-impedance state and can be jumped to either polarity on the fabricated section of the I/O board. For programming and debugging convenience, all handshake output lines can be read back. This implementation lets the user develop a variety of custom I/O designs. Most tristate I/O devices should be able to interface to this bus without added support chips. If desired, several devices can be accessed simultaneously (e.g.,

including a BOO4 evaluation board, meets the need for a coherent prototyping environment. The TDS Occam compiler supports the creation of processes that use "channels" for communication. These soft channels are mapped into physical links when a program is configured. Given this capability, programs intended for a particular interconnection scheme using a specified number of nodes can be simulated on variant networks, or even a single Transputer, if desired.

Evolution of a Board Design

The Yale XP/DCS CPU design (Figure 2) provides a Transputer-based node on a single board. It can be customized to handle data acquisition and I/O by adding a matched daughterboard to standardize the interface between the computational network and the physical world. Key features include a T800 processor, 128-Kbyte (zero-wait-state) SRAM, and a fiber-optic interface link for harsh environments. The design was standardized to the Single Extended Eurocard form factor. The rear edge connector is pin-compatible with the Immos ITEM system evaluation cards.

Figure 1: The 1.5-MFLOPS (sustained), 32-bit Transputer is fine-tuned for interprocessor communications. In robotic motion control, this proves a major plus.
two 16-bit or three 10-bit D/A or A/D converters could be accessed by attaching the various chips to different parts of the I/O bus.)

In asynchronous I/O mode, devices can be accessed independent of speed. A complete I/O cycle with three I/O bus accesses in this default mode would take less than 1 μsec, or roughly the time for one floating-point operation. Thus, I/O timing considerations are negligible compared to computational requirements. Typically, the three external memory cycles are: enable device via handshaking output, read from/write to device (if needed, after delay), and disable device. However, if a faster I/O cycle time is required, the board allows for direct bus interfacing to the Transputer via a synchronous I/O mode. This is possible by removing bidirectional bus latches.

**Going for the Juggler**

The first application of the XP/DCS board set was to control a planar juggling apparatus (Figure 3a). It was built to learn more about the underlying principles in modeling, analyzing, and controlling robots that repetitively catch, throw, hop, run—or juggle. The physical system consists of a puck that slides on an inclined plane and is batted successively by a simple "robot" that consists of a bar with a cushion rotating in the juggling plane. It is driven by a dc servo motor from PMI Motion Technologies (Commack, NY). Figure 3b, a schematic representation of the juggling application, shows the concurrent processes operating on each processor.

To move the bar according to some puck-dependent control algorithm, the puck's position and velocity in both directions on the plane must be measured. This is done by placing an oscillator inside the puck and burying a sensing grid in the juggling plane, thus imitating a big digitizing table. The first XP/DCS node in the system functions as a smart sensor. In the tracking mode, it measures the induced grid voltages in a 10" × 10" window around the puck and computes the position from moment calculations. This information is used to track the puck along its flight, as well as to feed state observers in order to estimate velocities and reduce measured noise in position data. Each measurement of four puck states is communicated asynchronously via fiber optics to the motor controller. This process is performed at a rate of 1 kHz, which is at least 16 times faster than would be possible if a standard video camera with a vision system were used to acquire the puck states. The Occam code excerpt in Figure 4 shows the sensor program's structural components.

Parameters passed to this sensor procedure consist of the link specification to the controller, desired sampling time, state variable estimator poles, and juggling board angle. Following library includes and declarations is the procedure that scans one window around a given puck position. The main program consists of two parallel processes: a puck-tracking program and a communication program (device driver) that communicates with the controller (Figure 4).

This parallel construct serves two purposes. First, it exploits the fact that a Transputer can simultaneously communicate on several links without seriously affecting ongoing computation.

![Figure 3: (a) Controlling a planar juggling apparatus. The system consists of a puck, which slides on an inclined plane and is batted successively by a simple "robot" that consists of a bar with a cushion rotating in the juggling plane. The robot is driven by a dc servo motor. (b) Represented here are the concurrent processes operating on each processor.](image)

![Figure 4: This Occam excerpt shows the sensor program's structural components. Parameters passed to this procedure are: link specification to controller, desired sampling time, state variable estimator poles, and juggling board angle.](image)
Second, it decouples the timing of controller and sensor, allowing them to operate at different cycle times. If the sensor communicated directly with the controller, without intermediate device drivers, both would run at the same cycle time because, in Occam, communication enforces process synchronization. Communication on both sides of a transaction will only proceed upon completion. In this case the sensor performs a complete tracking cycle in a 1-microsecond interval that is roughly twice the controller sampling time.

These two parallel processes form the first process in a PRI PAR construct, making them both high-priority processes. High-priority processes have two main features: Their timers have 1-μsec ticks as opposed to 64-μsec ticks of low-priority processes, and they have priority over low-priority processes. A high-priority process cannot be interrupted by a low-priority process. Thus, in a parallel construct, one has to make sure that a high-priority process will not run continuously and exclude other processes from running. Naturally, processes that handle link communications should run in high priority, as external communication should be serviced promptly. The tracking program needs to run at high priority only because of microsecond timing requirements for I/O handling.

Figure 5 shows an Occam mode excerpt that depicts the structure of the program running on the controller processor. Again, the program consists of a parallel structure with two high-priority device drivers and the low-priority juggle control program. The two device drivers—one for communication between the B004 and controller for logging, and one between the controller and sensor—decouple the cycle times on the different processors and exploit communication/computation parallelism.

Several tasks are performed in the control process (Figure 6). Sampling time synchronization ensures a constant, fixed sampling time of 640 μsec. Motor position and velocity update tasks include reading the optical encoder, deriving and filtering motor velocity, and performing safety checks. Puck states are read from the sensor node, and safety checks are made. A robot-control algorithm is implemented that specifies a desired motor trajectory; a high-gain PD feedback motor control is performed to achieve it. Another task is to detect the puck status. Finally, online logging to the B004 in the host PC is performed.

```plaintext
{{

F CONTROLLER
PROC CONTROLLER(Chan ofAny fromSENS, toSENS, VAL BOOL, log, 
VAL INT, rows, interleaves, misc, 
VAL REAL32, smpltime, boardangle, deg, alpha, x.d, ydot.d, 
set.k1, set.k2, set.k3, 
incratio, Kp, KV, gravtorque, offset_torque)

... includes and declarations

PRI PAR
PAR

... sensor--controller device driver

... controller--logger device driver

... juggle control

}}
```

**Figure 5:** The main program consists of two parallel processes: a puck-tracking program and a communication program (device driver) that communicates with the controller.

```
{{{

juggle control
initializations

WHILE NOT abort
SEQ

time.update()
theta.omega.update() read.puck.states() robot.continuous()
puck.phases() online.log()

shut.down.motor() terminate.logging() flash(abort, mask)
}}
```

**Figure 6:** Several tasks are performed in the control process. These include sampling time synchronization, motor position, and velocity updates. Also, puck states are read from the sensor node, and safety checks ensue.

Empirically verified, the computational speed for the 20-MHz T800 Transputer is 1.5 sustained MFLOPS. This figure, of course, does not include external memory accesses required for much of the application code. For example, a single-precision operation $a = b + c$, with variables in external memory (zero wait states), typically requires 1.3 μsec. This is slower than the Transputer's claimed floating-point performance as it includes three external memory cycles. Longer expressions where the floating-point unit can work off the evaluation stack without storing intermediate results in memory occur more quickly. The one-line excerpt from a controller computation (Figure 7) was timed between 6.6 and 8.8 μsec, depending on data. All the variables are REAL32, except hit dist, which is a constant (VAL REAL32).

```
result := ((x * y) - (x . dot . y) + 
(sign . x * hit . dist + (x . dot . y) . div1)) . div2
```

**Figure 7:** This controller computation excerpt was timed between 6.6 and 8.8 μsec, depending on data. Six multiplications, two divisions, and three additions are involved.

External I/O speed and interrupt latency are of concern for real-time control applications. Running an Occam test program for both showed that I/O accesses are essentially external memory cycles. The timing using the PORT construct for one I/O operation was measured as 250 nsec. Typical (worst-case) interrupt latency is 19 (78) cycles (assuming use of on-chip RAM), which for a 20-MHz T800 translates into 0.95 cycles (3.9-nsec) depending on the length of the presently executed instruction.

For parallel robotic chores, the Transputer offers many appealing features. Prime among its assets is interprocessor connectivity. Versus large parallel systems, as well as bus-based units, the Transputer-based two-board set represents a "best fit."