BBN Advanced Computers Inc.

TC2000™ TECHNICAL PRODUCT SUMMARY

TC2000™ Product Summary

Revision: 2.0

November 1989

BBN Advanced Computers Inc.
10 Fawcett Street
Cambridge, MA 02138
(617) 873-6000

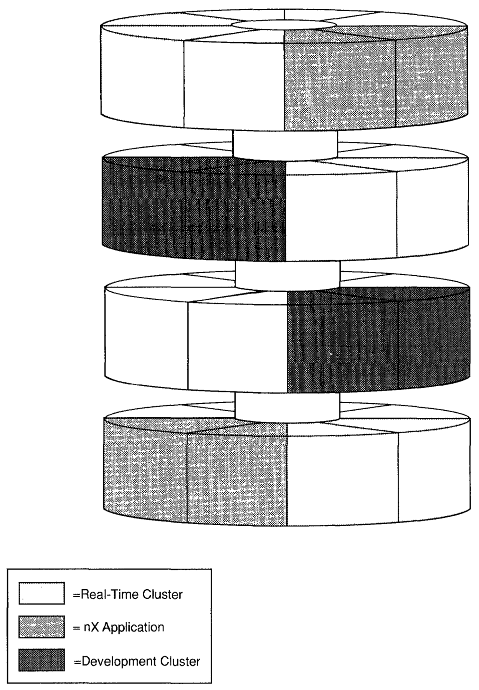
Notice

Trademarks

Table of Contents

Introduction

Today's computing applications bring with them a level of complexity and a demand for raw power barely dreamed of a decade ago. Nowhere is this trend more pronounced than in time-critical systems where the system must respond to real-time events. This job generally involves coordinating many existing components that were never designed to function with each other. As a result, the system designer must work with a diverse range of components with widely varying operational characteristics. In aerospace simulation, for example, operating systems, languages, and even fundamental time-scales may be different for each piece of equipment, yet most or all of the hardware must interact, and overall high computing power or throughput is a must. On-line transaction processing, another time-critical task, requires rapid memory access, high availability, high throughput, high disk performance, and uncompromised database updates. Both applications involve many activities taking place at the same time, of durations lasting from microseconds to seconds, and require both compliance with emerging standards and the availability of development tools for tuning system performance.



Managing these systems today, and improving upon them for tomorrow, calls for a single computing system that can truly integrate all levels of hardware and software into one flexible, manageable environment. Time-critical applications call for the TC2000™ system.

Time-critical problems contain a number of characteristics that challenge the traditional high-performance computer, and yet map well to the TC2000 system:

- Predictable response to many events
- High aggregate computational power
- Predictable performance
- High systems throughput
- Large data sets
- Interdependence among events and data
- High availability and reliability

The TC2000 system offers the perfect solution for time-critical software development and execution on all these counts, because it can start small for prototyping, then grow to precisely the amount of processing speed, I/O, and memory needed for the final application. Unlike other time-critical, high-performance systems, the TC2000 concept lets the user adjust the system to fit current and future needs.

A SOLUTION FOR THE 90'S

In today's computing applications, moving from design concept to finished system is a major undertaking. Both the size of the final software package and the length of time required for its development are difficult to predict. The result is that hardware requirements frequently change as software development progresses. What appeared to be a perfectly adequate system at the outset may prove to have inadequate memory resources or insufficient processing power as software development nears completion. Any project manager has seen efforts to shoehorn software onto a machine as the project deadline nears. The result is usually a software package that does less than originally intended, and has problems in documentation and maintenance resulting from the last-minute fixes and compromises.

These problems exist in conventional systems because the only practical way to increase processing power is to upgrade to a larger computer. Whether the solution has been to make do with the first machine and compromise on the software, or to trade up to something more powerful, the result has been one of two situations, both undesirable: either the computer is fully utilized and slightly too small for the application, or it is oversized for the application and excess resources are wasted.

BBN engineers have long believed they could design an economical computer that is both sufficiently expandable and configurable to satisfy all aspects of time-critical needs. Accordingly, BBN Advanced Computers has built the TC2000 system, which delivers affordable supercomputer power from multiple processors, in a system that is easily configured and expanded to meet the user's needs. By combining the development, input/output, and multiple execution environments into a single framework of shared and dedicated resources, the TC2000 computer can:

- Manage total system complexity
- Scale to meet changing requirements
- Minimize development time and cost
- Reduce cost of ownership
- Maximize availability and reliability

All of these attributes permit time-critical applications of many types to map readily onto the TC2000 design.

THE TC2000 DESIGN

Built around the Motorola 88000 RISC processor family, the TC2000 computer expands from eight processing elements to a tightly integrated system containing as many as 504 processors. It delivers over 9500 MIPS of raw processing power (based on the Dhrystone benchmark), and runs floating-point operations at up to 10,000 peak megaflops. With tools such as BBN's Uniform System™ application library, the user can increase the amount of computing power applied to an application at any time without rewriting software.

Each microprocessor is combined with various amounts of DRAM memory, two cache memory management units, and optional VME I/O ports, onto a single printed-circuit board called a function card (currently either a TC/FPV or a TC/FP, with or without VME, respectively). Connecting all function cards is the unique processor/memory interconnection structure known as the Butterfly® switch. This hardware provides efficient and transparent access by each processor to all locations in memory, whether local to the processor, remote on another function board, or external on a disk or tape. Due to the modularity of the TC2000 system, interconnection bandwidth actually increases as more function cards and switch cards are added.

Memory

For every running process, users can designate sections of the memory to be private or shared, depending on the application requirements. This feature allows users to protect certain values in memory while allowing other users and other programs to access commonly needed values. Physical memory, like processing power, has a wide range of expansion possibilities, starting with 44 megabytes on an eight-processor system and increasing to one gigabyte on a 63-processor system, with an architectural limit of 16 gigabytes on a 504-processor system. A fully configured TC2000 system can hold extremely large databases in on-board memory, enabling a combination of speed and size in database applications that was never before possible.

Clustering

Another TC2000 advantage is its cluster design, a feature that allows users to allocate the resources of individual function cards into named groups, which may or may not be accessible by other users or other programs. A major benefit of this feature is that users can designate one or the other of the two TC2000 operating systems for a given cluster: nX™, for general-purpose software development based on the UNIX® 4.3BSD™ operating system, and pSOS^{+m}™, a time-critical, multi-tasking, multiprocessing executive from Software Components Group, Inc. pSOS^{+m} implements the Real-Time Executive Interface Definition (RTEID) specification.[1] The dual operating systems provide a spectrum of environments for the systems builder, including:

- pSOS^{+m} time-critical clusters, wherein a cluster is dedicated to a specific time-critical application
- nX dedicated clusters, wherein a cluster is dedicated to specific nX time-critical application processes
- the nX timesharing cluster for conventional, and multiprocessor, timesharing or software development

For example, the user can execute an application that uses pSOS^{+m} time-critical multiprocessor clusters to collect data, monitor setpoints, and control a device, while dedicated nX clusters support on-line data analysis, parallel simulation, or operator interfaces. At the same time other users can monitor the system or develop new code in the nX timesharing cluster. Users can develop (compile, edit, etc.) their time-critical programs under nX, then execute them under pSOS^{+m}, without encountering the complications of cross-compiling from a different platform. In addition, programs running under both operating systems can readily communicate through shared memory, a feature that greatly simplifies, for example, linking the outputs from various time-critical components to the inputs of complex simulations.

[1] Real -Time Executive Interface Definition Specification, MVMERTEID/D1, Motorola Inc.

Languages

Language compilers for the TC2000 computer have been optimized with time-critical applications in mind. Programmers who use Fortran on time-critical and scientific projects will find that TC2000 Fortran provides a familiar environment that offers extensions for efficient parallel programming. The TC2000 C compiler shares optimizations with the Fortran compiler and allows use of the Uniform System parallel run-time environment for convenient processor and memory management. Ada on the TC2000 system is Telesoft's TeleGen2™ second generation Ada compiler, aimed at applications with government and military requirements or reusable software in mind. Lastly, to extract the highest performance from inner-loop computations, programmers will find that the Motorola RISC processor is practical to program in assembly language.

Development Tools

A major feature of the TC2000 system is its rich set of development tools, optimized for programming in a multiprocessing environment. In addition to standard and enhanced UNIX tools, BBN offers the Xtra™ toolset. In its initial release, the Xtra package offers the TotalView™ source-level, multi-processing debugger and the Gist™ performance analyzer. TotalView, which supports both the nX and pSOS^{+m} operating systems, was designed from the ground up to go beyond conventional debugging packages and fully exploit the interactive features of the X Window System™. By opening a window for each running process, TotalView lets users see the effects of changes in one process on one or more other processes. The Gist performance analyzer is a graphical tool for examining the performance of multiprocess programs. Gist supplies the programmer with a picture of the dynamic interactions of processes running in parallel. With this global view, programmers can quickly identify areas worthy of program tuning, and characterize the effects of such external factors as operating system overhead.

I/O and Peripherals

The TC2000 system's input/output capacity is as flexible as its processing capacity, offering excellent opportunities for designing time-critical systems where sensors and other devices require very high speed communications. Because of the unique TC2000 architecture, total system bandwidth actually increases as components are added, rather than being further divided among additional components as in conventional multiprocessor systems. Thus the TC2000 computer can deliver aggregate I/O bandwidth ranging from eight megabytes per second to 320 megabytes per second, with an architectural limit of over 4000 megabytes per second.[2]

Each TC/FPV function card in a TC2000 system provides an independent industry-standard VMEbus™. Every system has at least one TC/FPV to support the standard peripherals, but systems can be configured with additional cards to support applications with high aggregate I/O bandwidth requirements. The interface conforms to IEEE 1014 and supports master, slave and system controller functions on the VMEbus. A 16-megabyte window into the VMEbus

[2] Each TC/FPV provides an independent VMEbus interface capable of operating at eight megabytes per second. The standard TC2000 VMEbus midplane supports five independent VMEbuses per eight function cards, so a 63-processor system can support 40 VMEbuses at eight megabytes per second each. See Chapter Seven for more details.

address space can be mapped into the TC2000 system global address space, providing I/O resource access to both the local processor on the function card and to other processors in the system. Similarly, devices on the VMEbus can access local memory on the TC/FPV to which they are attached and can also access memory on other TC2000 function cards throughout the system.

TC2000 peripherals include high-performance 8-inch Winchester disks, 5-1/4 - inch removable disks, 1/2" and 1/4" tape systems, asynchronous terminal controllers, and local area network interfaces. The TC2000 design adheres to industry standards for peripherals and communications, such as IEEE 802.3 (Ethernet™) and the SCSIstandard.

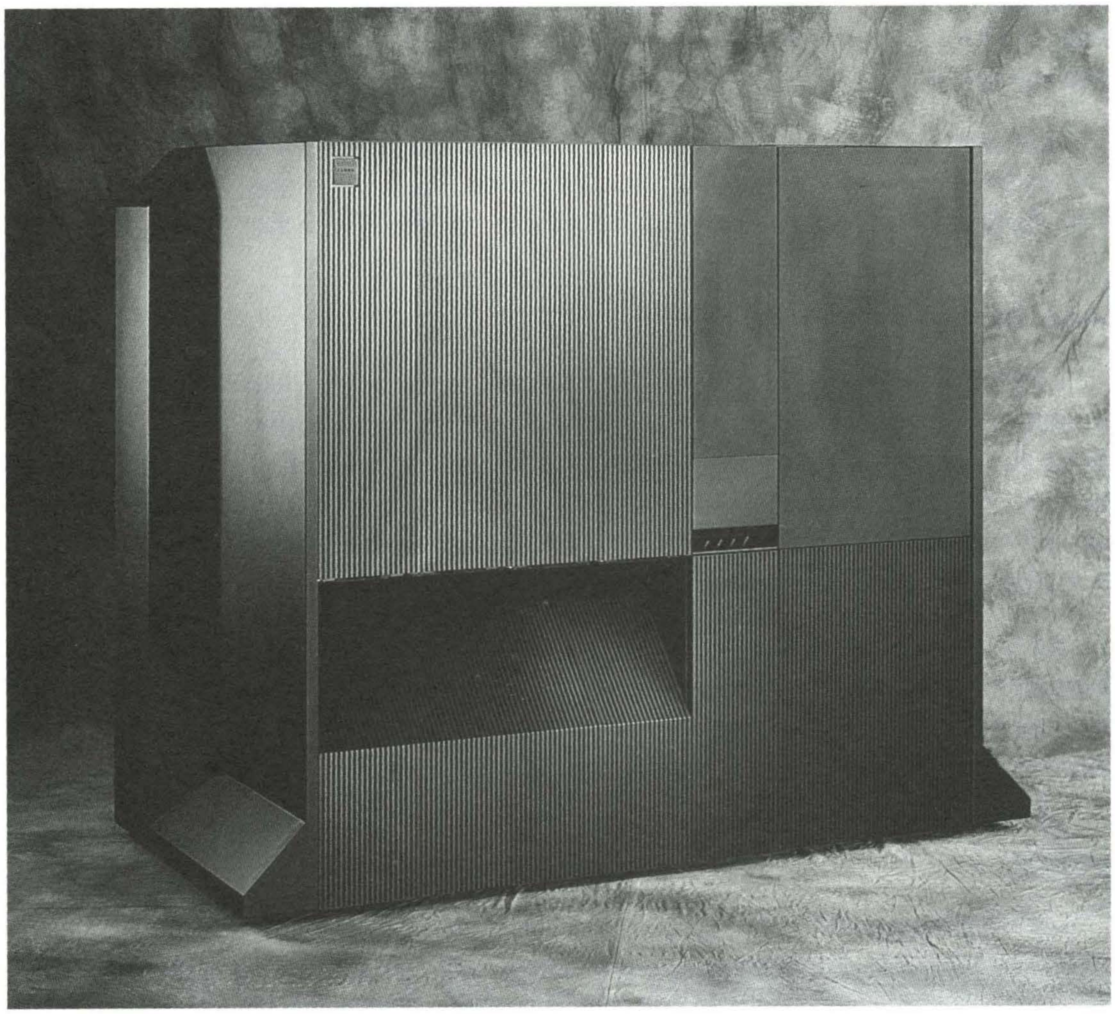
Test and Control System

The TC2000 computer offers improved reliability over conventional systems. A built-in Test and Control System (TCS) includes a separate master TCS processor, a slave single-chip microprocessor for each function card and pair of Butterfly switch cards in the system, and an independently powered and clocked diagnostic bus. Through the slave processors, the TCS monitors voltage and temperature levels, key components, and critical signals throughout the computer and logs to its own disk any problems found. Using the integrated TCS diagnostics and the menu-driven console interface, field service personnel can rapidly isolate problems to a field-replaceable unit.

The TCS takes faulty boards off-line for test and replacement while the rest of the system can be restarted and continue running independently. Thus users benefit from very short mean-time-to-repair (MTTR) and high system availability without paying extra for backup systems that may never be used. Built-in diagnostics and event-logging capabilities boost system availability and reliability while reducing the requirements for maintenance.

Packaging

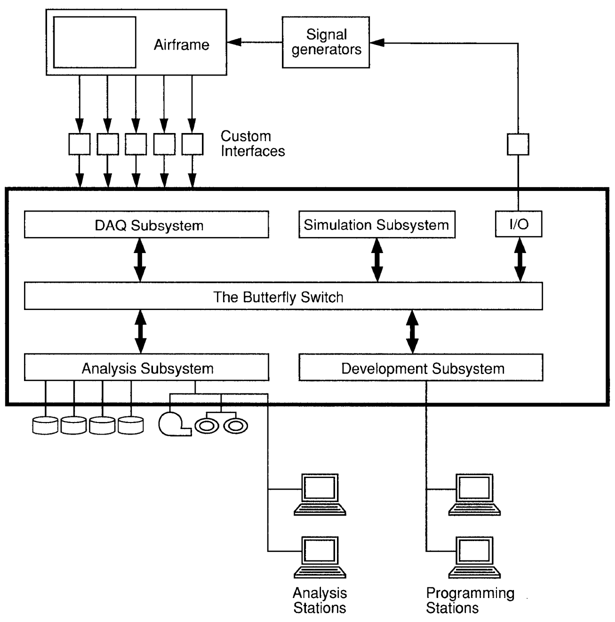
The architectural scalability of the TC2000 system is reflected in the packaging. A thirty-two processor TC2000 system, for instance, consists of a system base module and three expansion modules. The system base module forms the core of all TC2000 systems and consists of three units: the expansion module, the utility module, and the peripheral cabinet. TC2000 systems can currently be configured with up to seven additional expansion modules to meet users' computational, memory, and I/O requirements. An expansion module holds up to eight TC2000 function cards and includes two Butterfly switch modules, a power supply, and an optional eight-slot VME midplane and VME power supply. Expansion modules can be easily and quickly added to existing systems in the field to provide additional Butterfly switch capacity and function card slots. The inclusion of a separate power supply with each expansion module enables TC2000 systems to scale electrical power capabilities as the system grows, unlike some other systems in which the power system places limits on system configuration options. The system base module also includes a utility module and a peripheral cabinet. The utility module contains the TCS master processor (including TCS disk drives and modem), the system clock card, one 1/4-inch cartridge tape drive, one eight-inch Winchester disk drive, and the front panel controls for the system. The peripheral cabinet is used to house additional peripherals, including disk drives and 1/2-inch tape drives. The peripheral cabinet mounts standard 19-inch rack equipment. By adding additional expansion modules and peripheral cabinets, TC2000 systems can be expanded to add more processing, memory and I/O resources.



The 32-processor TC2000 system shown consists of a system base module, three additional expansion modules, and an additional peripheral cabinet.

Time-Critical Computing

Time-critical systems have always been capable of rapidly responding to events and meeting specific, quantifiable timing imperatives. The demands of the physical world impose hard deadlines; traditional time-critical systems meet these demands with dedicated application platforms, using disparate computers to support various components of the application. A flight simulator, for example, might have individual computers devoted to various functions such as the visual displays, flight controls, avionics, data recording, terrain data base, operator interface, and post run analysis. In commercial time-critical applications, the highest performance transaction processing systems might require specialized hardware and software systems to track inventory, make worldwide reservations, update branch-office databases, and respond quickly to hundreds of simultaneous users.



Time-critical systems have become increasingly complex in recent years. Today's system builder is faced with the traditional timing imperatives as well as requirements for unprecedented system performance and flexibility. These new requirements have grown out of the central role that time-critical computing is taking in the success of increasingly demanding technical and commercial endeavors. We will review these trends in time-critical systems, then show how the TC2000 system's uniquely integrated combination of scalability, shared memory, dual operating systems, clustering, and development tools provides the capabilities necessary for efficiently implementing time-critical systems.

- Responsiveness and predictability
- Many external and internal events
- High aggregate computational power requirements
- Simultaneous high-computation functions
- High event and data throughput
- Simultaneous multiple data flow
- Large data sets
- Interdependence among events and/or data sets
- High reliability and availability

The characteristics of time-critical computing are well matched to the TC2000 system. The pSOS^{+m} executive offers responsive, predictable performance. The system's scalable, shared memory, multiprocessor architecture can handle many interdependent events and large data sets. TC2000 multiprocessors and clusters are ideal for handling simultaneous computational functions. The switch-based multi-processor interconnect offers simultaneous data paths and inherently high reliability and availability

TRENDS IN TIME-CRITICAL COMPUTING

A number of recent trends in time-critical systems have pointed out the restrictions inherent in the traditional approach to building complex time-critical systems. Perhaps the most significant trend is the increasing complexity of the systems being simulated, monitored, and controlled (e.g. aircraft, spacecraft, weapons, and air traffic systems). These modern systems often have significant numbers of processors in their own right. Their increasing complexity leads directly to the need for more complex simulation and emulation systems.

The demand for increased performance to improve fidelity in time-critical systems is almost axiomatic across a wide range of applications from data acquisition and vehicle simulation, to discrete event simulation. Flight test, signal, and image processing systems have shown an increasingly voracious appetite for computer resources. For example, the desire for higher-fidelity modeling is leading to requirements of hundreds of megaflops in time-critical helicopter simulations. Rather than running complex blade dynamics models off-line, design engineers want to run the full engineering models in real-time as part of a man-in-the-loop simulator so they can get immediate feedback from simulator test pilots.

The increasing complexity and performance requirements of target systems also places increasing demands on data acquisition, analysis, and storage requirements. These trends have led to increasing requirements for input/output performance. This issue is exacerbated by the fact that single processor computing power has been increasing much faster than disk and tape storage performance.

- Simulation, control, and monitoring of increasingly complex systems
- Higher fidelity simulations and emulations
- More and faster data collection, storage, and retrieval
- Flexibility to support simultaneous simulations and multipurpose subsystems
- Integration of modeling applications (e.g. CFD, FEA) with programming models distinct from traditional purely frame-based models
- More use of simulation to replace "live" testing
- More use of emulation of subsystems for engineering evaluation
- Improved setup, operation, and analysis interfaces
- Standard computing and development environments

Trends in time-critical systems are placing increasing demands on the system implementer for performance and flexibility.

In addition to the increasing complexity of basic target systems, the need for simulating interactions of multiple subsystems is growing. Robert St. John, in a 1987 paper, says that the study of space-station buildup scenarios, the control of multiple free-flying bodies, and the support of multiple orbits "now demand computer hardware and software systems capable of supporting simultaneous simulations built upon modular, interchangeable, cost-effective systems."[1]

Advanced simulation systems require multi-purpose computing systems to support modeling of structures, fluid dynamics, and the operating environment of the target simulation. Large space-based structures, for example, cannot be rigid. Their designers have to be able to model their structural dynamics in real time as part of simulating these structures. Yet these modeling codes (e.g. finite element analysis, CFD, underwater sound transmission models) are not organized like traditional event-based and frame-based time-critical systems codes. Rather they are "scientific computing," and have typically been handled by scientific supercomputers even when the results were time critical. These engineers need a system that supports an appropriate execution environment for both types of code.

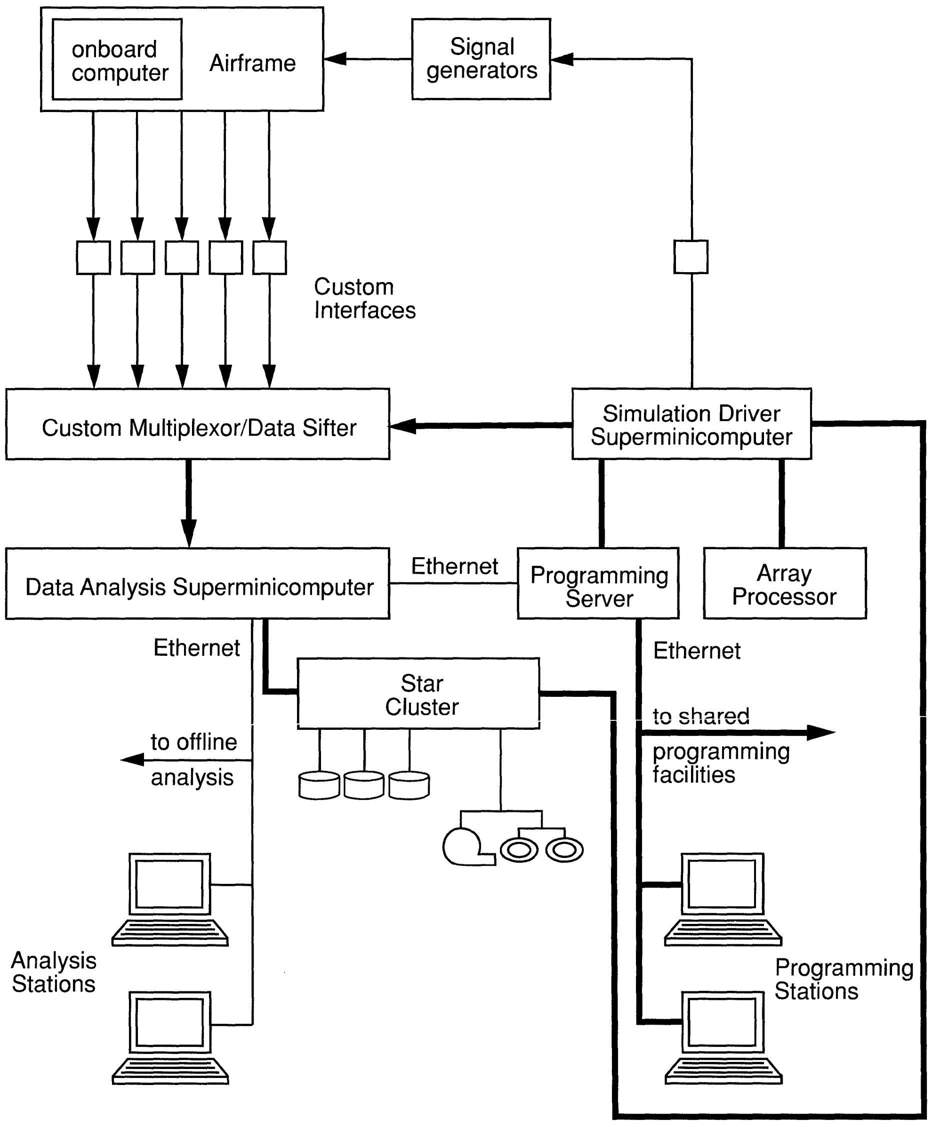
As the cost of testing real systems has increased, simulation of the operating environment of devices under test has been increasing. Traditionally, simulators have been used to provide training — especially for situations that would be risky. Now the decreasing cost of high fidelity simulation allows simulators to replace live trial runs in system development. The device under test is placed in the loop of an operational environment simulator and data acquisition system. This technique can be used in both development and production testing. Operational environment simulation offers lower cost, lower risk, and better instrumentation than is typically available in a live trial.

While time-critical systems are becoming more complex, they are also becoming more widely used — as in the design and analysis of aircraft or spacecraft. Thus the user community is changing. Simulators are no longer the exclusive tools of the simulator designers. The new users need simpler, and yet more powerful, interfaces to the system. "With complexity of the typical telemetry system increasing from year to year, device designers, system designers, and programmers are continually working on techniques to hide this complexity behind an operator-friendly man-machine interface."[2]

Another major trend in time-critical computing is the need for standards. As time-critical systems have become more complex and need to interact with other systems, the need for networking standards has increased. There is a growing need for standard environments to minimize development costs and make use of commercial software packages practical. Even in situations where suitable commercial tools are not available, a standard operating system makes it much easier to develop custom tools.

The time-critical systems builder has been responding to these trends with multi-processor systems made by connecting together conventional computer platforms. While this has been successful in the past, it has become increasingly difficult to keep up with the demands for increased system power and applications flexibility. Debugging is significantly more difficult on these systems than it is on single-processor systems. The connectivity approaches that were successful when small numbers of individual platforms were linked have become bottlenecks as more processors are brought to bear on an application. Furthermore, while isolated platforms do help maintain "firewalls" between subsystems — which is useful in subsystem development and debugging — the resulting total system is too often inflexible and difficult to scale as processing requirements change. Computing and input/output resources cannot be easily redeployed as application requirements change. The development tools used on one platform are seldom useful

[1] St. John, Robert H., et al, "Real-Time Simulation for Space Station," Proceedings of the IEEE, Volume 75, No. 3, March 1987, p 383.
[2] Strock, O.J. October 1988, "Trends in Telemetry Systems," Proceedings of the International Telemetering Conference, Volume XXIV, pp 1-3



The traditional approach to time-critical systems implementation has led to increasingly complex interconnections among independent computing platforms. Each computer must be separately programmed, debugged, and linked with the surrounding equipment. These interconnections increase the complexity of the system design and limit systems growth as they become bottlenecks. This approach produces boundaries that are rigid, making it difficult to reallocate computational, memory, and input/output resources as application requirements change. The resulting system is also difficult to debug since one can not obtain a total, dynamic view of the system with the tools available for the individual platforms.

in debugging another platform in the system. Even if the various platforms use the same debugging tools, obtaining a total, dynamic view of the system is difficult or impossible. Since much of the effort in such complex systems lies in software development and maintenance, dealing with these disparate machines can become a full-time problem. Finally, while such systems can provide high system availability through platform-level redundancy, it is an expensive solution.

THE TC2000 SYSTEM SOLUTION

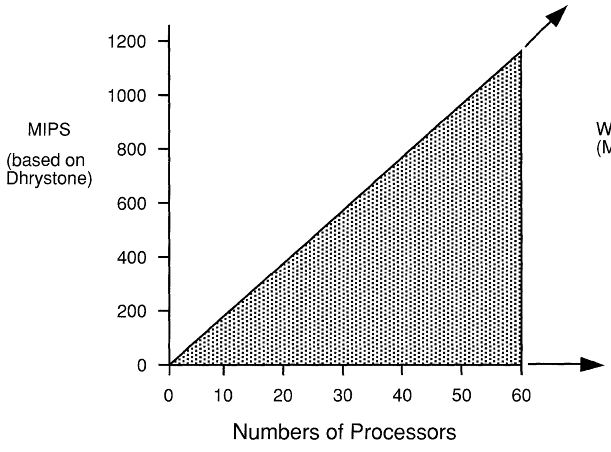
The TC2000 system helps today's system builder meet the increasingly complex requirements of time-critical systems by providing:

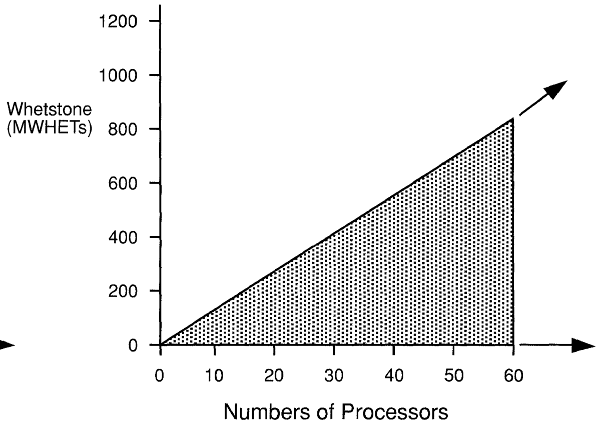
- Scalability in both computational and input/output performance
- Management of system complexity in all phases of a project from development and execution to maintenance
- Management of development time and cost with integrated languages, development tools, and standard operating systems
- High availability, reliability, and maintainability due to the system's switch-based multiprocessor architecture, reconfigurability, and attached test and control processor
- Low cost of ownership through the use of standard microprocessors, standard operating systems, and a truly integrated system from a single vendor.

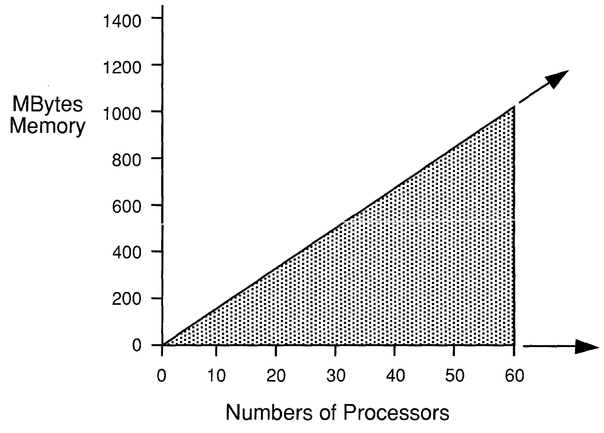
The TC2000 Butterfly switch forms the basis of the system's scalability. Scalability is important because applications come in many sizes and change over time. In a large development project with many subcontractors, each subcontractor might need a different-size system for development of their subsystem. The prime contractor will need a larger system capable of running all the subsystems. This is easy with the TC2000 system. Furthermore, the development tools used by the subcontractors will support debugging on the total integrated system. Or a group may be responsible for a wide range of simulators ranging from relatively fixed training simulators to simulation laboratories where flexibility to adapt to new projects is very important. In some projects, the development system might be larger than the deployed training systems. The TC2000 system's scalability and flexibility lets designers use the same basic system and tools over a wide range of projects, thus minimizing learning-curve costs.

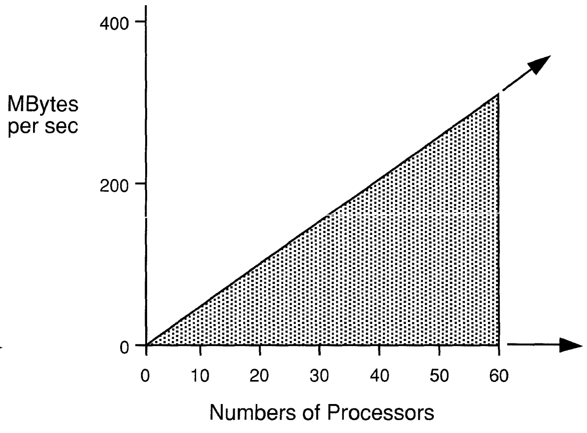
It is common to have contractual requirements to provide headroom to grow a system. It has been a truism that today's application system will be inadequate for tomorrow's needs. The TC2000 system is ready now to adapt to changing application requirements. Unlike traditional interconnection approaches — memory busses, multipoint networking, and point-to-point connections — the TC2000 switch-based shared memory avoids all of the single-path bottlenecks and increases in interconnection complexity as the number of processors increases.

The TC2000 system provides management of systems complexity through its integrated clustering, shared memory, and multiple operating systems. It features a unified system that can adapt to a wide range of system requirements. Over the course of a development project, the resources in a system might start out largely devoted to time-sharing use in the development cluster. As the project progresses, more and more resources can be devoted to debugging the run-time aspects of the application and input/output drivers. After the development is completed, the system's resources can be reconfigured with software commands when desirable for software maintenance.









The TC2000 system supports unprecedented scalability due to its switch-based shared memory system. The initial TC2000 release supports up to 63 processors and the switch architecture supports up to 504 processors. This assures the processing, memory, and I/O capabilities needed to meet changing application requirements over time.

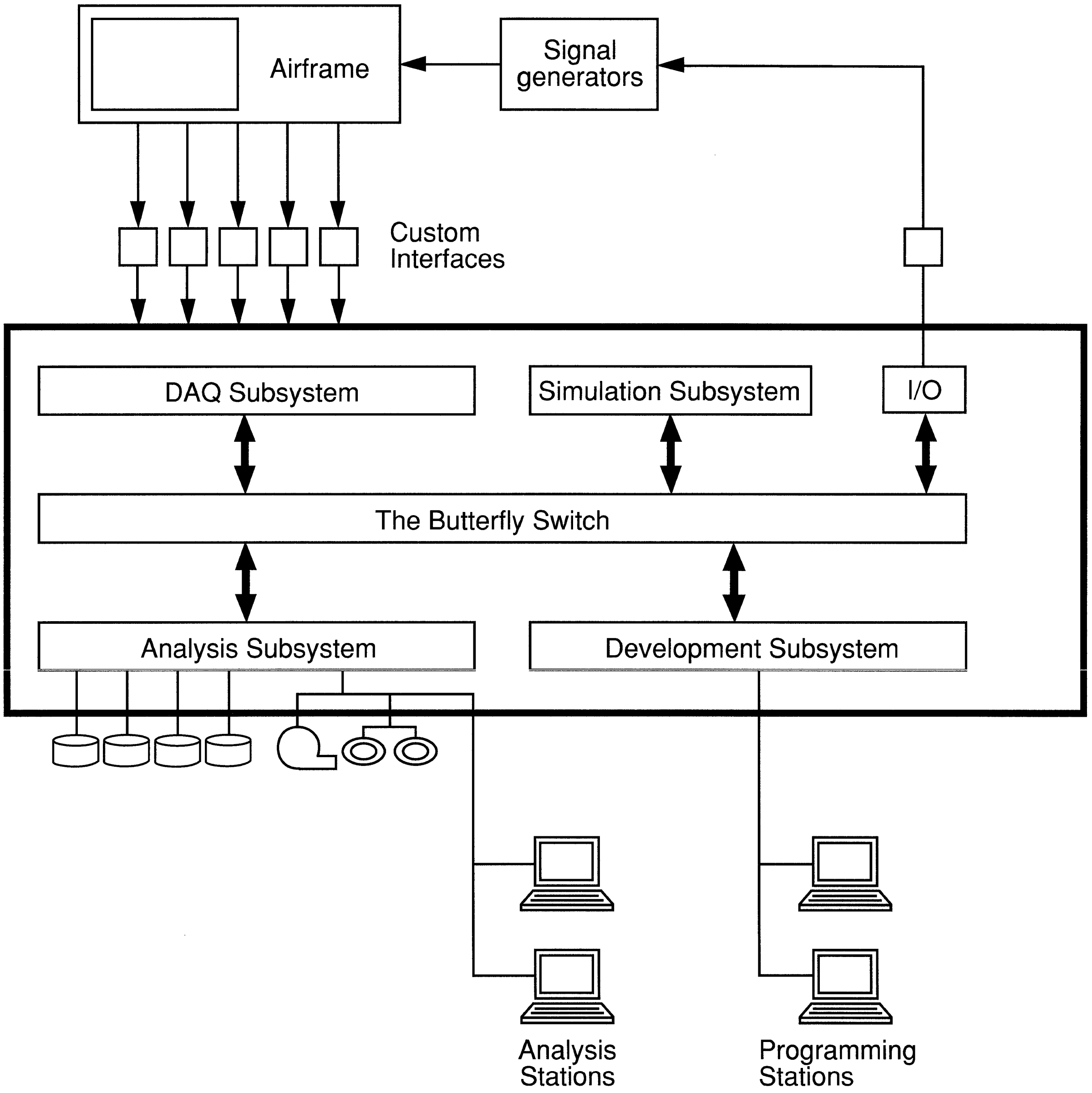
Where the traditional time-critical system design approach allocates parts of an application to specific isolated processors, the TC2000 design improves upon such "hard-wired" allocation by assigning multiprocessor groups — clusters — to an application or application subsystem. This allocation is dynamic and the resources can be reallocated or expanded as application needs change. As an example, you might have a particularly complex vehicle to simulate. Most of the system might be devoted to this during one session. In a later session you might want to simulate the interactions of a group of simpler vehicles. You can easily reconfigure the TC2000 system to support both applications. Unlike previous hardware platforms that users lashed together to build complex systems, the TC2000 architecture comprises a single, seamless computer environment capable of serving the present and future needs of an application.

Clustering provides an effective means to isolate components of the application, while shared memory provides the connectivity required in the overall system. Clusters have access control mechanisms similar to the nX file system. The owner of a cluster can restrict control and shared-memory access to the owning user, or to a specific group of users, or allow access to all users on the system. Clusters have names, and can persist beyond the lifetime of the nX process that created them.

Thus clustering serves as a means of supporting the traditional multitasking, multiprocess model of the UNIX operating systems, so that several users or applications can share a single TC2000 computer system. It also serves as a convenient means of supporting and modeling current paradigms of system design and development in which a system or application is decomposed into modules or objects. The TC2000 computer can serve simultaneously as development platform and test bed. TC2000 clustering provides a spectrum of environments on a single integrated platform and supports the flexibility to reassign resources as requirements change. TC2000 scalability offers the performance required for modeling of complex systems in real-time.

The TC2000 system helps manage development time and cost by providing a complete, integrated solution. The nX operating system provides a familiar, rich UNIX environment without compromising the time-critical execution environment provided by pSOS^{+m}. Shared memory extensions in nX, pSOS^{+m}, and TC2000 languages provide straight-forward communications within and between clusters. The TC2000 development environment provides graphic debugging and performance analysis tools based on the X Window System. The source-level debugger, TotalView, provides a state-of-the-art multiprocessor debugging tool. With the TotalView tool, the user can simultaneously debug processes in multiple clusters, running under different operating systems. The Gist graphic performance analysis tool provides quick insight into the dynamic behavior of applications. The TC2000 system helps reduce development costs by providing standard languages to minimize the users' learning curve and to maximize the users' ability to reuse code from other projects. The system also helps minimize integration and special development costs for input/output devices with a standard VMEbus interface.

By combining the inherent properties of a switch-based multiprocessor, the Test and Control System, and proven manufacturing methods, the TC2000 system provides high availability, reliability, and maintainability. Since the system is a switch-based design, the absence of individual processors does not prevent the system from running. Automatic configuration at boot time brings all working processors into the system. Service technicians, working remotely or locally with the TCS, can conduct rapid fault isolation. The TCS can then use its dedicated diagnostic bus to load and run diagnostics on a suspect card while the rebooted system is running applications.

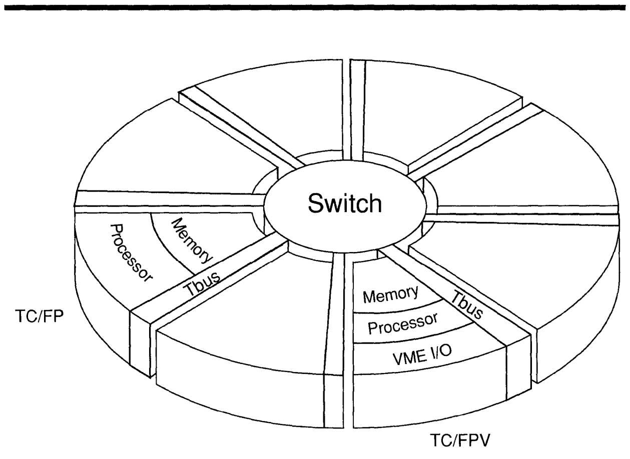


Where the traditional time-critical system design approach allocates parts of an application to specific isolated processors, the TC2000 design improves upon such "hard-wired" allocation by software-configurable assignment of multiprocessor groups — clusters — to an application or application subsystem. The processors within a cluster can communicate by shared memory. Similarly, clusters can communicate with other clusters by shared memory.

The TC2000 system also minimizes cost of ownership by using a standard, commercially available high-performance microprocessor. This assures that the system will stay on an aggressive price/performance curve as new technology becomes available. Since the system is an integrated set of hardware and software from one vendor, it eliminates the hidden costs of integrating and supporting disparate platforms. The flexibility offered by clustering means that users can allocate system resources in varying ways over time. In a conventional approach with a stand-alone development system, the development system might have to be oversized to assure good performance during peak development activities. The TC2000 system minimizes development cost by avoiding this oversizing of individual platforms; users can deploy TC2000 computing resources as needed at various phases of a project.

Function Cards

The keynote of TC2000 performance is scalability. The TC2000 modular system architecture makes it readily expandable and highly configurable thanks to a key system building-block, the function card. Each function card is the equivalent of a high-end supermini-computer capable of processing at 19 MIPS (based on the Dhrystone benchmark), 13 megaWhetstones, and 20 peak megaflops. The TC2000 system achieves its aggregate processing power and I/O capability by combining these function cards in a single system interconnected by the TC2000 switch.



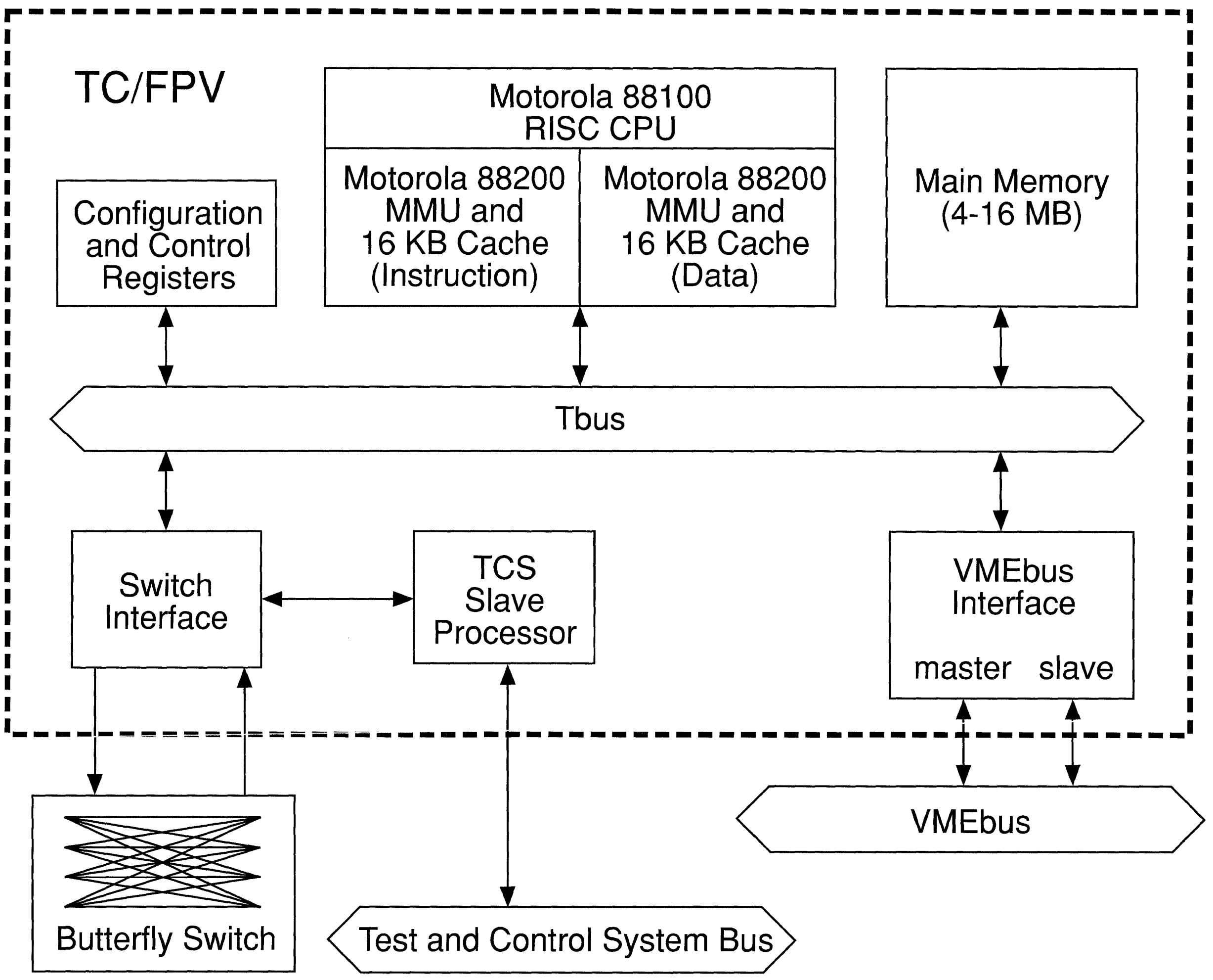
The TC2000 system supports multiple types of function cards, letting users tailor systems to their own application requirements. For example, the TC/FP function card includes an 88100 processor and 16 megabytes of memory, while the TC/FPV has an 88100, 4 or 16 megabytes of memory, and a VME I/O interface. Also on each TC2000 function card (connected by an on-board transaction bus), are two 88200 cache/memory management units (one for data and one for instructions), a switch interface, and a Test and Control System (TCS) slave processor.

For balanced applications, users can allocate TC/FP and TC/FPV cards in similar proportions. For applications requiring intensive I/O, users will want a preponderance of TC/FPVs, while compute-intensive applications would use a majority of TC/FP cards. Future cards will supply varying amounts of memory and different I/O interfaces as new standards evolve.

Function-card level modularity provides several system-wide advantages. Most obvious are its expandability and configurability. Equally important is the ability to upgrade the performance of individual system components as needed. Each TC2000 function card has its own independent clock, permitting independent processor speeds. As faster versions of the 88000 family become available, BBN will incorporate them into its function cards. Thus users will be able to add newer, faster technology as it becomes available without discarding their existing function cards.

THE MOTOROLA 88100 PROCESSOR

The TC2000 architecture uses a commercial processor rather than a BBN design. By using products from a large semiconductor manufacturer such as Motorola, we take advantage of their economies of scale and their extensive experience with fabrication. The high degree of integration such commercial devices deliver leads to smaller chip sets and compact system designs while drastically reducing the risk of system development. It also opens the door to third-party software support for the TC2000 computer. To that end, BBN Advanced Computers Inc. belongs to the 88open


TC/FPV
Motorola 88100 RISC CPU
Motorola 88200 MMU and 16 KB Cache (Instruction)
Motorola 88200 MMU and 16 KB Cache (Data)
Main Memory (4-16 MB)
Configuration and Control Registers
Tbus
Switch Interface
TCS Slave Processor
VMEbus Interface
master
slave
VMEbus
Butterfly Switch
Test and Control System Bus

TC2000 function cards provide the processing power, I/O and memory resources in TC2000 systems. Implemented on a single circuit board, each function card is equivalent to a high-end super-minicomputer.

Consortium, Ltd., which promotes the timely implementation of standards for hardware and software interfaces to the 88000.

BBN chose the Motorola 88100 processor because this chip represents the leading edge in current microprocessor technology. Based on a reduced instruction set computer (RISC) architecture, the 88100 includes its own IEEE 754 floating-point processing unit, as well as integer, data, and instruction units that operate concurrently.

THE ADVANTAGES OF THE MOTOROLA 88100 RISC PROCESSOR

As their name implies, RISC processors use a smaller instruction set than complex instruction set (CISC) processors. RISC processors employ a simple set of instructions that are optimized for speed and are implemented in hardware, rather than using microcode. By focusing on register-to-register operations and limiting memory-based operations to separate load and store instructions, RISC processors achieve extremely fast performance. The 88100 has only 51 instructions, compared to 130 or more on typical CISC processors.

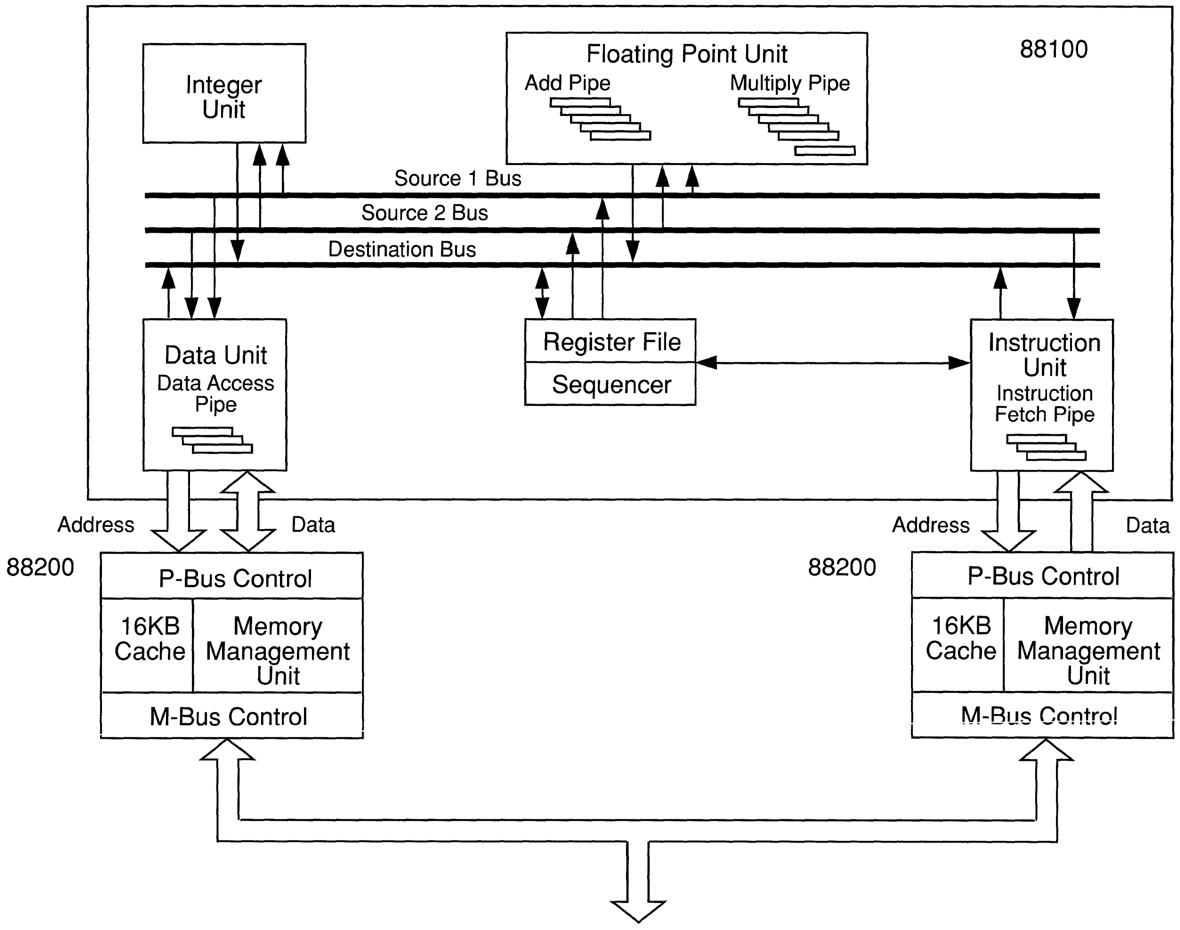
The goal of RISC processors is to execute an instruction every clock cycle. Because they use simplified addressing modes and employ simpler decoding mechanisms, RISC processors require fewer transistors than equivalent CISC processors and are therefore easier to implement in a given fabrication technology at a given performance level. Because of their simpler designs, they can also operate at faster clock rates than CISC equivalents.

The 16-KByte cache/memory management chips that support the data and instruction units on the 88200 make their operation as efficient as possible. The 88100 also gains efficiency by pipelining data, instructions, and floating-point operations, allowing one multicycle instruction to start on each clock cycle. Some of the internal efficiencies of the 88100's operation derive from the regular syntax of its instruction set, which has a fixed format for op codes and operands. BBN's compilers optimize instruction scheduling to take advantage of 88100 pipelines and delayed branching mechanisms.

To optimize the use of their architectural features, RISC processors use sophisticated optimizing compilers. With integer, floating-point, instruction-fetching, and data-fetching operations all operating simultaneously in a pipelining fashion, the compiler has to make sure that the operations occur in the right order. Other RISC processors force compilers to achieve this synchronization by inserting "no op" instructions at key points in the instruction stream, making it difficult to hand-code assembly language. Because the 88100 uses a scoreboard to keep track of synchronization instead of no-ops, it is easier to hand-code in assembly language for the 88100. Object code modules are also smaller without the non-productive no-op instructions.

Delayed branching lets programmers avoid some of the disadvantages of executing branches or program control structures in a pipelined system. Normally, branching interferes with the time savings achieved by pipelining data and instructions, because once a branch occurs, the remaining instructions and data already in the pipeline must be flushed. The refilling of the pipelines causes a slowdown. When delayed branching is specified, the next sequential instruction is executed before the branch instruction, regardless of the branch condition. So the pipeline continues to operate without unused cycles, and the next instruction in the pipeline executes while the branch target instruction is pre-fetched from memory. This provides an efficient use of processor resources when branches are taken because the time required to prefetch the target instruction is overlapped with useful instruction execution.

The assembly-language programmer can take advantage of the delayed branching feature by moving an instruction from its normal position before the branch to the delay slot after the branch, and specifying the delayed branch option for the branch.



The heart of each TC2000 function card is the first member of Motorola's newest and fastest processor architecture, the 88100 RISC processor. It combines a fast, 20-MHz clock rate, advanced CMOS technology and reduced instruction set computer (RISC) techniques. Two companion 88200 cache/memory management units provide separate instruction and data caches.

THE TRANSACTION BUS (Tbus)

The transaction bus (Tbus) connects the various subsystems on a function card. Running at 20 MHz, the Tbus is 32 bits wide, giving it a total bandwidth of 80 megabytes per second. Because the Tbus is a transaction bus, it supports split cycles, which prevents deadlock between systems on the bus and speeds communications by freeing the bus during long bus transactions.

Deadlock occurs when two devices on a conventional bus require data from each other at the same time. One device arbitrates the bus and requests data from the other, which is unable to respond because it is waiting for the bus to be free so it can request data itself. The Tbus permits one of the devices to send a signal that says, in effect, "I'll get back to you later," which permits the requesting device to release control of the bus. The device can then respond with data at a later time.

Split transactions are also used for long bus transactions. This allows other, faster bus transactions to take place during the longer split transaction.

THE SWITCH INTERFACE

Each TC2000 function card carries an interface to the Butterfly switch, implemented as a pair of custom CMOS gate arrays. The switch interface converts references to remote memory originating on the local Tbus into switch transactions, and converts incoming switch transactions into Tbus memory references. The interface then builds the packets that are transmitted through the switch and processes all protocols — such as checksums — associated with remote memory references.

The interface converts signal levels between the switch and the function card and also buffers data passing between the switch, with its clock rate of 38 MHz, and the Tbus, operating at 20 MHz. This decoupling between the switch clock and the function card clocks lets the system support processor cards that are running at different clock rates.

SPECIAL FEATURES FOR TIME-CRITICAL OPERATION

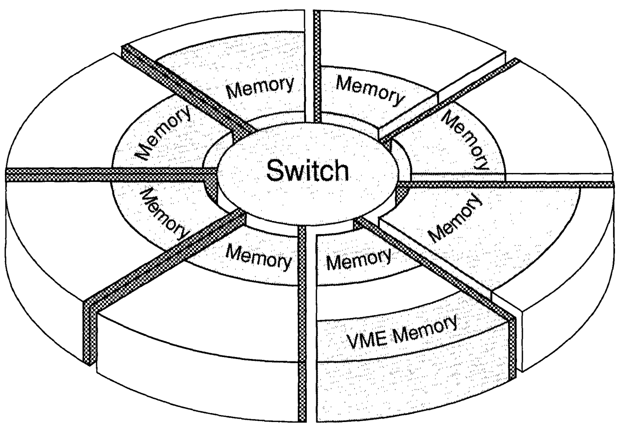
Each function card on a TC2000 has a real-time clock with a resolution of one microsecond. All real-time clocks in the system are globally synchronized to keep them operating within one microsecond of each other at all times, greatly simplifying applications that need global time stamps.

Each function card also has two 32-bit time-of-next-interrupt (TONI) registers, which are used by the nX and pSOS^{+m} operating systems. Under the pSOS^{+m} executive, the clock interrupt handlers can load these registers with the desired time of the next interrupt for use in situations where operations must occur at specific times or specific intervals. On function cards that are running pSOS^{+m}, one TONI register is always available for application program use.

To detect situations in which a remote-memory operation delays handling an interrupt, each function card has an interrupts pending timer with a resolution of 1 microsecond. This timer starts whenever an interrupt request arrives at the 88100. If the timer expires before the CPU recognizes the interrupt, the 88100 stops all data operations and saves the status in internal registers, aborting the remote operation. The 88100 then restarts the aborted operation after the interrupt service is completed. The value of the interrupt pending timer can be read, enabling application routines to measure the interval from an interrupt request to the service routine execution. Because the value of the interrupt pending timer is settable, the response times for the system can be bounded as needed for real-time response.

Memory

To support the high-performance Motorola 88100 processors, the TC2000 system features a powerful memory design composed of cache memory, local memory, shared memory, and VME memory. Each function card in a TC2000 system contains on-board cache and local memory, directly accessible by the processor on the function card. This distribution avoids the memory bottleneck common in bus-based systems and allows multiple memory references to occur in parallel. In addition to its own local memory, every processor can address and access memory on all other function cards in a TC2000 system through the Butterfly switch. Processors can reference memory on other function cards transparently, so the user can treat the system as having either a shared global memory or a distributed memory according to the demands of the application. The nX operating system has default memory-placement policies for automatically placing data and other objects within the memory hierarchy, but the user has the option of fine-tuning these policies using system calls.



All physical memory in TC2000 systems resides on the function cards. The TC/FP supports 16 megabytes of memory and the TC/FPV supports 4 or 16 megabytes. All memory is mounted on single in-line memory modules (SIMMs) for easy expansion and upgrade.

MEMORY MODEL

Each process under the nX operating system and each processor under the pSOS^{+m} operating system has its own four-gigabyte virtual-memory address space. nX reserves one gigabyte for system use, and leaves the remaining three gigabytes for the user process. All pSOS^{+m} tasks on the same processor share the same virtual-address space.

Through hardware on each function card, 32-bit virtual addresses are translated into 34-bit system physical addresses, giving the user access to 16 gigabytes of physical memory. The physical memory that is mapped into a process's virtual-address space can be on the same function card as the process (local memory) or can be located on another function card (remote memory). Remote memory accesses occur transparently through the switch. Both nX and pSOS^{+m} execute program code from local memory. nX supports demand-paging of process virtual-address spaces, allowing allocated virtual memory to be larger than the available physical memory. To keep overhead low and achieve predictable performance, pSOS^{+m} does not page the memory in pSOS^{+m} processors. Thus all code and data addressed by a pSOS^{+m} task must be in physical memory.

Under nX, memory can be mapped either private or shared. For example, data in physical memory that is accessed by only one nX process will be mapped as private to that process. Since it is not mapped into the virtual address spaces of other processes, they will not be able to access this data. Alternatively, memory can be mapped as shared, in which case the same physical memory is mapped into the virtual address spaces of two or more processes, enabling them all to access the same data. Shared memory can be either local (e.g., several processes on the same processor accessing shared local data) or remote (e.g., several processes on different processors accessing shared data through the switch). Under pSOS^{+m}, all local memory on a function card is accessible to all tasks on that card. Remote memory can be mapped into the pSOS^{+m} processor's virtual-address space to provide shared memory between pSOS^{+m} processors.

Memory can also be mapped as either cachable or non-cachable, indicating whether or not the data is eligible to be stored in the cache memory on the function cards. Private data and read-only, shared data, such as program text, stack, and process-private data, are typically mapped as cachable, while read-write, shared data is normally marked non-cachable. The system provides a set of calls to help users override the default cache management policies of the operating system, allowing shared data to be cached. For example, many frame-based applications can cache shared data during the frame and use system calls to synchronize the cache memories with physical memory at the end of each frame.

A 16-megabyte window into the address space of each VMEbus can be mapped into the system physical address space, enabling all processors to access all VME devices and VME memory. Similarly, VME devices can access both local and remote TC2000 memory through a similar 16-megabyte window into the TC2000 address space. TC2000 processors can map VME memory either private or shared, but VME memory is always mapped non-cachable.

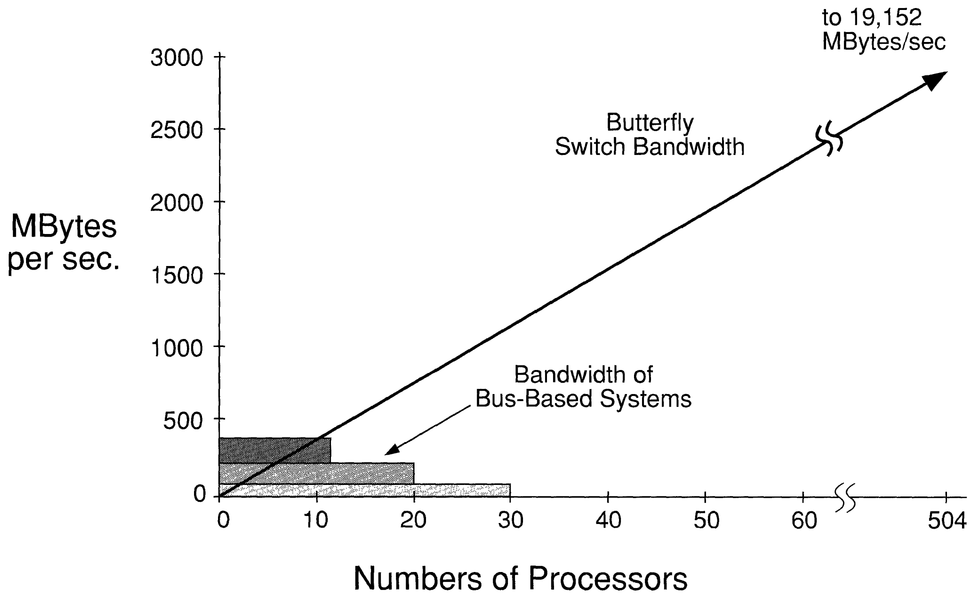
CACHE MEMORY

Cache memory provides a cost-effective way to speed overall system throughput. By using a small amount of extremely fast memory for the cache, the system maintains high performance while reducing the overall memory cost. When a processor needs to read or write data, hardware checks to see if the data is contained in the cache. If so, it is written or retrieved without requiring an access to system memory. If the data is not in the cache, it is written or retrieved from system memory and stored in the cache to speed subsequent accesses. By holding the most recently used memory locations in cache, a very large proportion of a typical program's memory references can be satisfied from the cache at high speed.

Memory management is supported in hardware by two Motorola 88200 cache/memory management units on each function card and by special function card mapping hardware. Each 88200 contains a 56-entry, fully associative page-address translation cache, maintained by the 88200 hardware, that holds the most recent memory mappings. Each function card has separate 88200 units for instructions and data, allowing instruction and data address translation to proceed in parallel. The 88200 maps 32-bit virtual addresses into 32-bit physical addresses; high-speed function card mapping hardware further maps the addresses into 34-bit system physical addresses.

Each cache line in an 88200 is 16 bytes long and any cache misses always result in a fill of the complete cache line. If the 88200 cannot find a particular word in the cache, it will load the 16 bytes of memory containing the word into the cache. This procedure can greatly speed execution, particularly for instruction references, since referencing sequential memory locations is a very common memory-access pattern. By filling an instruction cache line on each cache miss, even programs with long sections of sequential code will have an instruction cache hit-rate of at least 75%. Since cache lines are burst-filled from local memory, execution speed is improved in the sequential access case by reducing overall memory-access time. Cachable remote data is also burst-filled in 16-byte blocks through the switch.

Each processor maintains consistency between cache and system memory by one of two methods: write-through, which updates system memory whenever cache is written, and copy-back, which reconciles cache and system memory only when the cache line is flushed. The default method is copy-back, which typically provides higher performance, but the user can select write-through by means of a system call.



Bus-based systems that support larger numbers of processors typically provide lower per-processor bandwidth than smaller bus-based systems. The bandwidth of the Butterfly switch, in contrast, grows as the system size grows.

THE BUTTERFLY SWITCH

The Butterfly switch is the key component that provides the TC2000 system with scalable bandwidth capable of supporting multiple memory accesses at the same time. In bus-based systems, bandwidth is fixed by the bus and the user pays the full cost of that bandwidth up front when purchasing the system. As the user begins to add processors, the added processors increase contention for the bus. Thus in bus-based systems, the bus loses effective bandwidth with additional processors.

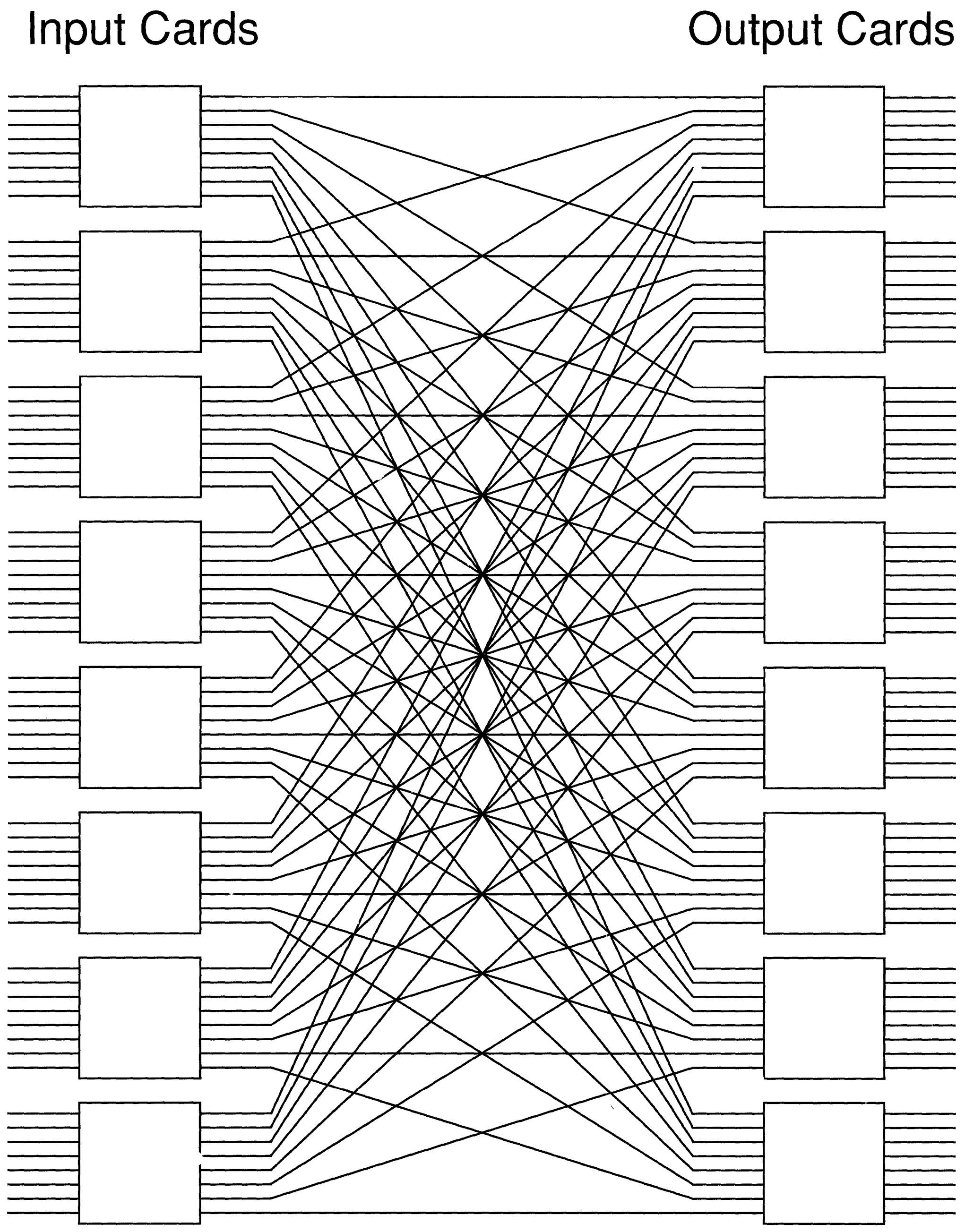
With the TC2000 design, system bandwidth is expandable. Because each TC2000 function card has its own switch interface, adding a processor adds another communication path and increases the total system bandwidth. The Butterfly switch also lets many different memory accesses occur simultaneously, which is not possible on bus-based systems. Thus a TC2000 system always maintains a balance among memory, processors, and bandwidth. Purchasing a TC2000 system guarantees the user an ideal path to adding system bandwidth later without having to pay for it all up front.

Each path through the switch in the TC2000 system is eight bits wide and is clocked at 38 MHz, providing a maximum of 38 megabytes per second per path of bandwidth. Adding processors increases the aggregate bandwidth by adding switch paths, up to a maximum aggregate bandwidth of 2,394 megabytes per second for a 63-processor system and an architectural maximum aggregate bandwidth of 19,152 megabytes per second for a 504-processor system.

The Butterfly switch uses multiple switching stages to provide a cost-effective implementation. Each switch module in the TC2000 system is an eight-by-eight crossbar and can connect any input to any output. Unlike a bus, which can only support one connection at a time, a crossbar can support multiple, simultaneous connections; each Butterfly switch module can support up to eight simultaneous connections. Very large crossbars could be used to interconnect processors and memory, but the cost of a crossbar grows as the square of the number of connections and quickly becomes prohibitive. The cost of a multi-stage network grows much more slowly, providing a cost-effective means to interconnect large numbers of processors. A two-stage TC2000 Butterfly switch has up to 64 ports while a three-stage switch can have up to 512 ports.

Each TC2000 function card contains a switch interface — a pair of custom gate arrays — that converts processor remote memory references appearing on the Tbus into switch messages, and performs all protocol and checksum processing associated with remote-memory references. The switch routes each (switch) message through the stages in the switch until it reaches its destination function card. At the destination function card, the switch interface converts the switch message into a Tbus memory reference, receives the results of the memory reference, and returns this result in a switch message back to the originating function card over the same switch path. The originating switch interface processes the reply and places the result on the Tbus for return to the processor. From the processor's point of view, a remote-memory reference looks just like a local memory reference but slower. This makes remote-memory references transparent to the user and enables users to refer to variables and data and manipulate them identically regardless of their physical location.

Fast and efficient message routing uses a serial-decision network scheme. Each message contains three bits of routing information for each stage in the switch (six bits for a two-stage switch and nine bits for a three-stage switch).



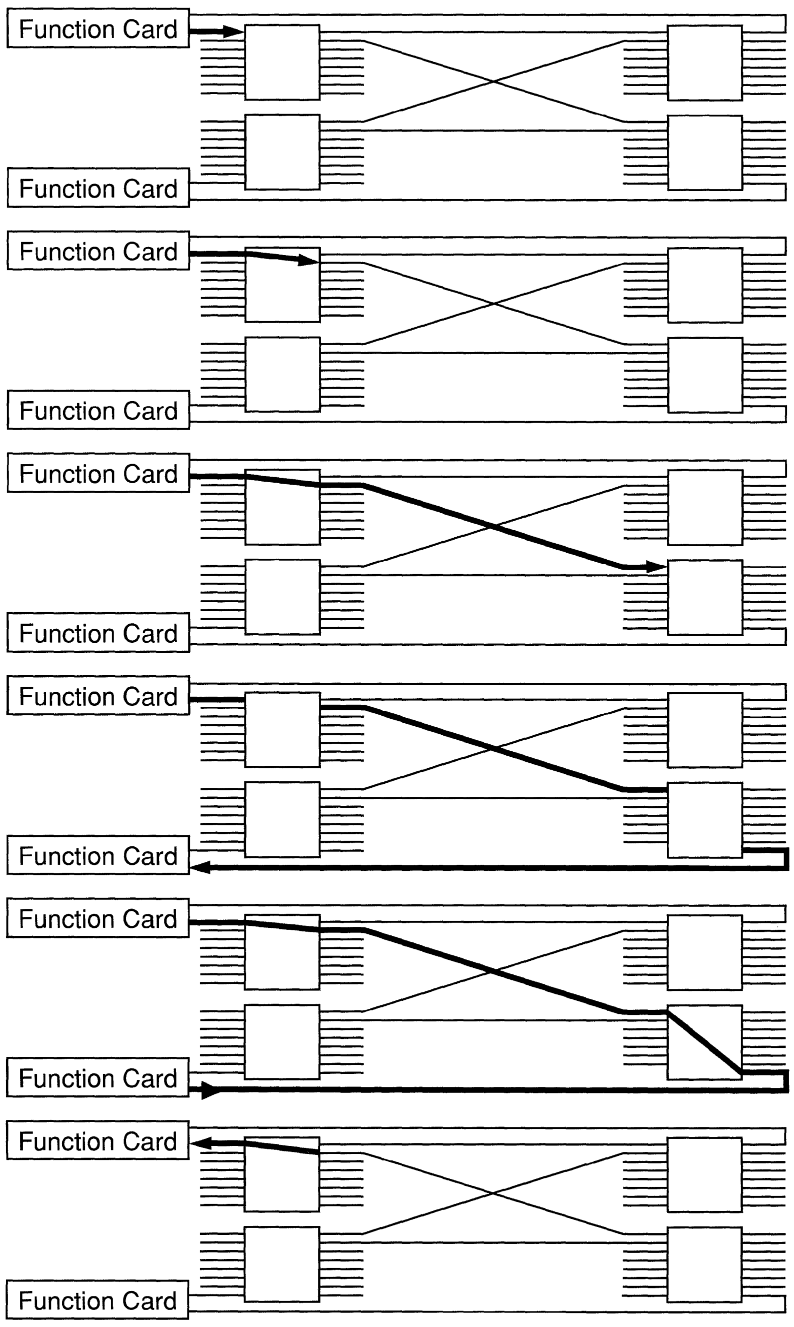
A two-stage Butterfly switch is made up of interconnected 8-by-8 crossbar modules and supports up to 63 function cards. Its modular implementation enables system bandwidth to grow with the number of function cards.

These three bits select one of the eight output ports on a switch module. As a message proceeds through the multi-stage switch, these routing address bits are used to route the message rapidly from one switch module to the next until the message reaches its destination.

Occasionally, two remote memory references from different processors may "collide" by trying to access the same remote-memory module at the same time. This is similar to bus contention in that one reference has to wait until the other finishes. By distributing shared data-structures and arrays across the memory in multiple function cards, TC2000 systems support multiple, parallel accesses to these data structures and arrays, reducing contention. When contention does occur, the switch module will select one of the references to complete first. The other message will be rejected, and the originating switch interface will automatically try it again. The TC2000 Butterfly switch also includes message slotting mechanisms that prevent processors from being locked out of a remote memory module and provide a bound on remote-memory access times during high levels of contention.

SYNCHRONIZATION

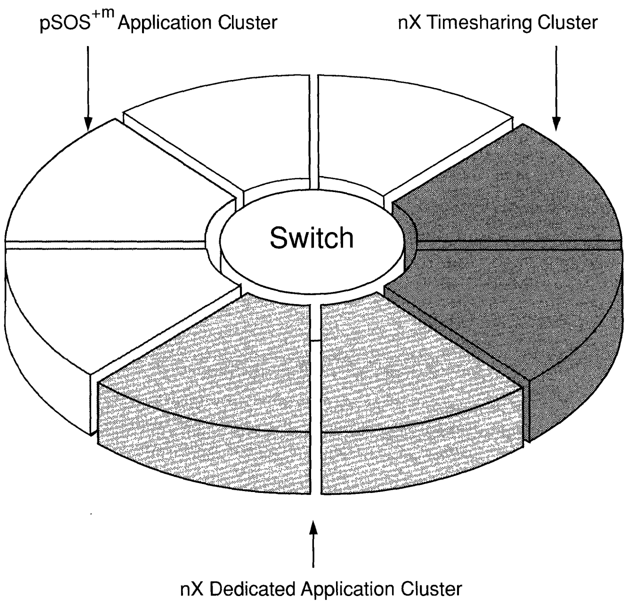
The TC2000 memory system includes synchronization mechanisms to allow programs running on multiple processors to coordinate and synchronize their operations and accesses to shared data. The fastest and most basic synchronization operation is the Motorola 88100 *xmem* (exchange memory) instruction. This simple operation indivisibly exchanges the contents of a memory location with the contents of an 88100 register. *xmem* can be used to implement more complex locking protocols. The TC2000 design supports the *xmem* operation for both local and remote memory.



The Butterfly switch routes each message through the stages in the switch until the message reaches its destination function card. The switch returns the result to the originating function card over the same switch path.

Operating Systems

The TC2000 computer offers two operating systems that can function concurrently in different clusters on the machine. The nX operating system is a general-purpose multi-user system based on the Berkeley UNIX 4.3BSD operating system. pSOS^{+m} is a real-time executive from Software Components Group, Inc. nX provides the TC2000 system's application development environment, an enhanced multiprocessor execution environment, and system administration, configuration, and booting capabilities. pSOS^{+m} offers the speed, responsiveness, and predictability needed for time-critical applications such as real-time data collection or simulation. Both operating systems take advantage of the TC2000 system's multiprocessing capabilities and its ability to cluster groups of processors together to work on an application.


pSOS^{+m} Application Cluster
nX Timesharing Cluster
Switch
nX Dedicated Application Cluster

DUAL OPERATING SYSTEMS

nX and pSOS^{+m} are tightly coupled on the TC2000 machine, communicating through shared memory. This marriage of a UNIX-based operating system to a fast, efficient, real-time executive offers the best of both environments. In particular, real-time executives are designed for fast, predictable response, and, on any given processor, will always outperform UNIX operating systems, even those implementations with real-time extensions. A time-critical data acquisition system, for example, can run on a pSOS^{+m} cluster, gathering data over the VMEbus and storing it in shared memory, while a data analysis module runs on an nX cluster, simultaneously analyzing the data.

Most importantly, the TC2000 system eliminates the usual problems of cross-development for time-critical applications by letting the user code and debug in a homogeneous environment where the same tools are integrated for both operating systems. Instead of using complicated porting procedures to deliver the application to its target once it is developed, followed by an even more complicated debugging process, users can develop and debug simultaneously on a single system.

For applications that require both high-level functionality and responsiveness, users can avoid the difficulties of trying to connect several different vendors' components. The TC2000 system provides a readily configurable environment that communicates internally through shared memory and externally through standard networks such as Ethernet, using TCP/IP and NFS protocols.

THE nX OPERATING SYSTEM

The nX operating system builds upon the familiar capabilities of the Berkeley UNIX 4.3BSD operating system. Like the Berkeley version, nX is a multi-user, virtual-memory system. In nX, however, the basic capabilities of the UNIX operating systems support time-critical applications in a dual-operating system, multiprocessing environment. Multiple copies of nX run on each processor node, taking advantage of local memory and multiple I/O paths. nX also complies with the IEEE 1003.1 POSIX interface specification for true UNIX applications portability.

nX Clustering

nX lets users group TC2000 function cards into clusters that match the application's computational, memory, and I/O requirements. Clusters have an access control mechanism similar to the TC2000 file system; the owner of a cluster can designate that cluster as permissable for use by the owner, by a group of users, or by all users. Users can create new clusters, and disband or change the size of existing clusters. Users can define clusters of free function cards to hold resources until they are needed and to serve as a place to return free cards. Users can also name clusters, and can let them persist beyond the nX process that created them.

Processor clustering lets users dedicate system resources to an application or to parts of an application. Using the clustering model, users can run various applications in different execution environments on the same machine. One group of function cards, for example, can form a "public" cluster for time-shared usage, while another cluster executes time-critical applications under pSOS^{+m}, and still another works on a private nX application. The size and number of clusters is determined entirely by the user, up to a limit of 64 clusters per system. Fifty-nine of the 64 clusters are available for users; the system reserves the remaining five, as follows:

- system cluster (a "super cluster" of all function cards in the system)
- free cluster for nX function cards
- free cluster for bare function cards (pSOS^{+m})
- nX I/O cluster
- nX public cluster

nX Interconnectivity

In addition to the standard networking components of UNIX operating systems, such as TCP/IP and Ethernet, nX supports the new layered products that offer standard application-level protocols for connecting to other devices and systems within a distributed workstation environment. These layered products include Sun Microsystems' Network File System™(NFS) and the X Window System developed at MIT for graphic user interfaces.

NFS is a distributed networking protocol that lets a user work with remote files and directories as though they were on the user's own machine. Included with NFS is Yellow Pages™, a network administration program that allows a single password file to support a multi-system, multi-vendor heterogeneous network. NFS and Yellow Pages make it easy to use the TC2000 system in a workstation-oriented environment. A TC2000 machine can serve as a central resource for computing while the various workstations connected to it provide a friendly, window-based user environment.

The X Window System is an important element of that environment. An industry-standard windowing system, the X Window System gives the TC2000 computer transparent access to workstations and other window-based display devices across a network. The X Window System supplies graphics primitives for menus, mouse control, and window management to help users develop portable windowing interfaces for their own applications. The TotalView debugger and Gist performance analyzer also use the X Window System.

nX Virtual Memory Management

Because nX is a virtual-memory demand-paged system, the effective range of memory locations available to a process is independent of the amount of physical memory actually present in the TC2000 computer.

nX divides the virtual-address space into eight-kilobyte pages, which are fetched from disk into memory when a process references a virtual-memory location that is not currently memory resident. nX pages out certain pages, moving them from memory to disk to clear memory space for new pages. This lets a machine or cluster run larger programs than it could normally run, given the amount of physical memory available.

The Berkeley UNIX replacement policy removes the pages that have been least recently used from memory. nX uses this policy as a default, but it allows the programmer to designate pages of virtual memory to be immune to page-out. These wired-down pages can be shared among processors or clusters, and can be accessed as named objects. Thus the programer can make sure that critical code or data stays in physical memory, eliminating the longer execution time that memory paging can cause.

The programmer also has control over the page placement policy under nX. Unlike the Berkeley UNIX system, which does not allow the user to designate the physical location of a virtual-memory page or control virtual-memory placement policy, nX lets the user allocate a page of virtual memory to a specific cluster, a specific processor, or a specific physical memory address.

nX also gives each process an independent, three-gigabyte user virtual-address space, compared with the two-gigabyte virtual-address space available from the Berkeley UNIX 4.3 system.

nX supports a disk cache buffer in main memory which reduces the number of times the system has to access the disk. Minimizing disk access is particularly important in parallel programs, where copies of the same text are executed in multiple processes, and in applications that generate large intermediate files, such as compilers. Users can configure the amount of memory in the disk cache and can distribute the memory in any manner over the system nodes.

nX as a Multiprocessing Environment

The nX user can reap the benefits of multiprocessing without explicitly using multiprocessing commands within an application, because nX builds multiprocessing into many standard functions of the UNIX operating systems. UNIX multitasking, for example, has been extended in nX to let the operating system automatically assign newly created processes to different processors, using dynamic load-balancing.

nX also incorporates a number of standard UNIX utilities that have been modified to take advantage of parallelism. *pmake*, a parallel version of the standard update utility make, distributes its work across multiple processors, helping users significantly reduce the time required to build an executable version of a complex program.

nX also helps the programmer make explicit use of multiprocessing within an application. Because many programs are modular at the task level, they lend themselves quite naturally to simultaneous execution. nX can break up a discrete-event simulation program, for example, into tasks modeling the behavior of the various subsystems being simulated. The programmer can use the *fork_and_bind system* call to assign newly created tasks to specific processor nodes, applying multiprocessing power directly to the application. When combined with clustering, *fork_and_bind* lets the user dedicate a processor node to a specific task or application, guaranteeing uninterrupted use of the processor's full computational bandwidth.

nX Shared Memory Management

Shared memory is key to effective cooperation among the TC2000 system's multiple processors. Less advanced operating systems let only related processes share memory, through forking or inheritance. The *vm_mapmem* system call lets unrelated TC2000 processes share memory, by allowing a process to map a file for shared access. Data placed in a shared-memory file then becomes directly accessible to all nX processors in the system, without the need for explicit copying. Clusters running nX can even map memory from clusters running pSOS^{+m}.

Synchronization between processors, provided by the 88000 *xmem* instruction, protects the integrity of this shared data. *xmem* allows programmers to synchronize access to data structures, ensuring that each thread of a computation can access a data structure only when the data is in a consistent state.

The *user_sleep_and_unlock* call and the *user_wakeup* call implement blocking semaphores. These calls provide the most efficient way to synchronize access to shared memory if the application is going to wait for more than a few seconds.

nX Process Scheduling

nX extends the capabilities of the UNIX scheduler to support time-critical applications. The Berkeley UNIX 4.3BSD scheduler, designed for a time-sharing environment, provides a priority-based scheduler that decides in a "fair" manner which process to execute next. This scheduling policy is generally unsuitable for time-critical environments. nX gives the programmer control some over the scheduler. The programmer can delay scheduler preemption of a process, thereby preventing competing processes from interrupting critical regions of code.

THE pSOS^{+m} REAL-TIME EXECUTIVE

A lean, but fully functional, multi-tasking, multiprocessor executive, pSOS^{+m} provides fast, predictable performance for time-critical applications. This interrupt-driven, real-time executive is optimized for applications demanding maximum performance and predictability. It also supports user-written interrupt handlers for devices on the TC/FPV VMEbus. pSOS^{+m} is compliant with the Real-Time Executive Interface Definition (RTEID).

Multiprocessing within a pSOS^{+m} Cluster

pSOS^{+m} provides multiprocessing support within a single pSOS^{+m} cluster by:

- Sharing names of pSOS^{+m} objects within a pSOS^{+m} cluster
- Implementing remote pSOS^{+m} system calls between processors in a pSOS^{+m} cluster

For example, while a pSOS^{+m} message queue is a processor-local object, its name can be exported to other processors within the pSOS^{+m} cluster. So while the create queue system service call originates on the processor that actually contains the queue, the queue name and cluster-wide ID can be exported to other processors within the cluster. Tasks on other processors can then import the queue ID by giving the queue name to the identify queue call. With this queue ID, tasks anywhere in the cluster can send messages to or receive messages from the queue. Other system objects — such as tasks, buffer partitions, and semaphores — support multiprocessor activity the same way.

Interrupt Handlers

Users can write nested interrupt handlers in pSOS^{+m}. These interrupt handlers, using certain system service calls, let the application control the execution of tasks in response to interrupt events. A user-written interrupt service routine, for example, can make a queue send call to post a buffer to a device read queue. If the task waiting on the queue has a higher priority than the task that was interrupted, pSOS^{+m} will context switch from the interrupted task to the higher priority task after the interrupt handler completes.

System Service Calls

System service calls within pSOS^{+m} are organized into eight groups:

- Task management
- Storage allocation
- Message queues
- Events and asynchronous signals
- Semaphores
- Time management and timers
- Fatal error handling
- Device driver services

Task Management

pSOS^{+m} supports multi-tasking with a priority-based, preemptive scheduler. The scheduler, which supports up to 255 user task priorities, always runs the highest priority ready-to-run task. It can also schedule tasks by round-robin time slicing on a per-task basis. A given task can execute in supervisor mode or with interrupts disabled if required; this flexibility supports virtually any real-time programming model.

Storage Allocation

pSOS^{+m} allocates memory from user-specified, physically contiguous sections of memory called regions. Typically regions have distinctive attributes. One region might consist of shared memory, for example, while another might contain the application's private memory. Users can construct variable-length, contiguous memory segments from a region. Since segments have variable lengths, a heap-management algorithm is used to manage segments within a region. Within a segment, users can construct multiple partitions for fixed-length buffers. The buffers within a given partition have fixed lengths, so their allocation and freeing is faster than the same operations would be for variable-length segments.

Message Queues

Message queues provide a mechanism for queuing fixed-length (16-byte) messages. They handle larger messages by passing the address of buffers or other user-defined objects. Messages can be assigned to message queues on the same or other processors within the pSOS^{+m} cluster. A task may poll for messages on a queue, or suspend execution (allowing other tasks of lower priority to run) until a message arrives. If multiple tasks are waiting on the same queue they can wait in either first-in-first-out (FIFO) order or in order of task priority. A task can also specify that it will wait until a message arrives or until a timeout occurs.

Events and Asynchronous Signals

Events provide a mechanism for synchronously indicating the occurrence of an event to a task. Thirty-two events can be used. Sixteen are reserved for system use; the other 16 are user-definable. A task can signal events to tasks on other processors within the cluster. A task must make an explicit system call to receive an event. The receiving task can poll for events or suspend execution while waiting for a particular "and/or" combination of events. An optional timeout can be specified to resume execution of the task if the specified events have not occurred by the timeout.

Asynchronous signals may be sent to tasks that have defined an asynchronous signal routine (ASR). Unlike events, for which the target task must explicitly make a system call to receive the event, asynchronous signals may arrive at a task at any time. If the task is running, the ASR is immediately entered. If the task is not running, the ASR will be entered the next time the task is scheduled to run. From the tasks' point of view, asynchronous signals look like software interrupts and the ASR is an interrupt handler. Thirty-two signals can be sent; 16 are reserved for system use. Asynchronous signals can be sent between processors within a pSOS^{+m} cluster.

Semaphores

pSOS^{+m} implements semaphores which tasks can use to synchronize execution and arbitrate shared resources. When created, an initial count specifies the number of tokens available for the controlled resource. A task can poll to acquire a semaphore or suspend execution while waiting for a semaphore. If multiple tasks are waiting for a semaphore, they can wait in either FIFO order or in order of task priority. Tasks may also specify a timeout period to resume execution if a semaphore does not become available within the specified interval.

Time Management and Timers

pSOS^{+m} supports a calendar date and time mechanism as well as task pausing and timed events. BBN's default clock interrupt handler sets up a processor card real-time clock and calls *tm_tick* at the rate specified in the user-provided cluster configuration file. The pSOS^{+m} time and date are set to the nX time and date when a cluster is initialized.

In addition to the basic calendar timer functions, pSOS^{+m} tasks can pause for a specified interval or until a specific absolute time. Events can also be sent to a task after a specified interval or at a specific absolute time.

Fatal Error Handling

The user application may pass control to a fatal error handler if it discovers a non-recoverable error. If the pROBE^{+}debugger is being used, the user application passes control to it. Otherwise, a user-specified callout is checked. If the application has provided a fatal error handler, it is executed. The system call can also signal a cluster-wide shutdown, resulting in a fatal error handler being called on all processors in the cluster.

Device Driver Services

The pSOS^{+m} executive provides a standard system call interface to support devices. If the user adheres to this standard interface, user I/O devices are accessible via pREP/C^{+} input/output function calls. Otherwise, the user can define a different, more specialized interface.

Interprocess Communication Channel Manager

The TC2000 Interprocess Communication Channel Manager (ICCM) permits communication and synchronization between processes running under different operating systems as well as between processes running within the same operating system. An Interprocess Communication Channel (ICC) is a shared memory object that passes fixed-length message buffers. The length of the buffers for a given ICC is specified when the ICC is created. To be as efficient as possible, ICC operations are typically executed at user level without requiring a trap into the kernel. ICC operations need the kernel's help only when delivering a message to a process that has blocked waiting to receive a message from an ICC. ICC buffers are not actually copied from one process, or processor, to another. Rather, shared memory is used to map the buffer area into the address spaces of the communicating processes.

Shared Memory

TC2000 pSOS^{+m} memory can be shared both within clusters and between clusters. Memory mapped by a pSOS^{+m} application must be physically located within a pSOS^{+m} cluster. All of the memory addressed by a pSOS^{+m} application is wired-down. This preserves fast and predictable performance for the pSOS^{+m} application.

A pSOS^{+m} application has access to the physical memory within its own pSOS^{+m} cluster. This application memory is divided on a function-card-by-function-card basis between code space, private data, and shared data. Private data is initially mapped so as to be accessible only by code on the same function card. Shared data is mapped into the address space of all of the function cards in the cluster. The cluster configuration file specifies the size of the private and shared regions for each processor.

TC2000 memory can be shared between pSOS^{+m} clusters by using the system named-object management mechanism to bind names to specific shared-memory objects. An object must be exported by the creating cluster to make it accessible to other clusters.

The system named-object management mechanism provides the means for binding nX virtual memory to specific pSOS^{+m} memory objects. An nX process can map pSOS^{+m} cluster memory into its virtual-address space. This shared memory appears as wired-down memory to the nX process, since the memory must be physically located within a pSOS^{+m} cluster. A pSOS^{+m} cluster that has memory mapped by nX processes cannot be disbanded until all of the nX processes mapping the memory directly, or indirectly, unmap it. This prevents dangling memory references. Access to memory objects is controlled by cluster-access privileges.

nX File System Support

I/O operations can be directed to user-supplied pSOS^{+m} device drivers, to local nX file systems, or to remote NFS-mounted file systems. Each pSOS^{+m} cluster is associated with an nX server which provides file system access through standard language input/output constructs.

Debugger Support

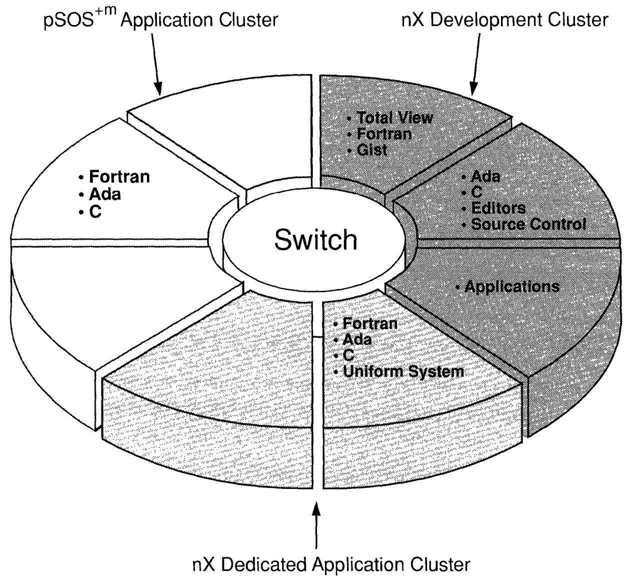
On the TC2000 System, pSOS^{+m} applications are fully supported by the TotalView debugger and Gist performance analyzer. However, BBN also supports Software Components Group's standard assembly-language level debugger and analyzer, pROBE^{+}. This package provides the following features:

- Conventional breakpointing, register dumps, and instruction disassembly and assembly
- pSOS^{+m} system call breakpoints
- Interactive system calls
- Run-time analysis

See the Programming Environment section for more information.

Programming Environment

The TC2000 system provides a complete integrated programming and development environment for time-critical applications, including all the software engineering features inherent in the UNIX operating systems, along with a full set of programming languages and compilers, including Fortran, Ada, C, and assembly language. Enhancements provided by nX — and accompanying tools such as Gist, TotalView, and the Uniform System — elevate the TC2000 package well above the standard UNIX operating systems as an ideal environment for software development. And since the development system is also the target system, the TC2000 user never encounters the unpleasant surprises that can arise from porting the code to the target.



Today's time-critical applications frequently require multiprocessing systems. Standard software development tools provide no help to the programmer in dealing with these issues; without the proper tools, the programmer's inability to spot undesired dynamic interactions and race conditions can become a major stumbling block.

The TC2000 programing environment addresses the special problems of multiprocessor and time-critical programming. It provides tools both for examining the state of all the processors involved in an application simultaneously, and for analyzing the dynamic behavior of many interacting processes. Because pSOS^{+m} and nX operate in a tightly coupled fashion, TC2000 development tools present an easy-to-understand view of the entire system, making the TC2000 computer unique among multiprocessing systems.

THE nX DEVELOPMENT ENVIRONMENT

Programmers must work efficiently and effectively to achieve the cost, time, and specification goals of software development projects. This is becoming increasingly difficult in today's complex time-critical applications environment. Time-critical applications are by definition performance-demanding, requiring execution in a dedicated environment. High-end applications have frequently required multiple target processors or computer systems.

The programming tool set for a development environment must reduce or eliminate the more mundane tasks of programming to let developers focus on the engineering challenges of the application. It must be easily extensible, to allow programmers to customize it. It must match the architecture of the target execution environment, even when the program will be distributed over multiple processors. Lastly, the tool set, and the attributes of the overall programming environment, must also be familiar to the development team, or at least easy to learn.

The TC2000 system provides a complete set of powerful tools for the development of time-critical software in and for a multiprocessing environment. TC2000 tools address all of the major programming tasks — coding, documentation, compilation, debugging, and performance tuning — and support a team approach to software development. The basic programming environment of the TC2000 computer is nX, which, because it is based upon the UNIX operating systems, forms a rich and familiar foundation for software development. For time-critical application development, the fact that nX co-resides with pSOS^{+m} on the TC2000 system presents major advantages over other approaches. Until now, developers either have had to purchase a separate cross-development host system, which can make development more difficult and integration problematic, or they have had to settle for time-critical implementations of the UNIX operating systems, which generally sacrifice time-critical responsiveness to support high-level functionality.

nX provides an extremely flexible and powerful user environment with features that support rapid prototyping. The shell, or command interpreter, for example, has a built-in high-level language of its own that allows the user to combine, or pipeline, programs without the need for compilation or recoding. nX offers the user a choice of two standard shells — the C shell and the Bourne shell.

A good editor is critical to the efficient creation of an application's code and accompanying documentation. nX includes the standard set of UNIX text editors (vi, ed, and ex), as well as the GNU Emacs screen editor. GNU Emacs provides an advanced set of capabilities that can be customized and extended by the user, with built-in help facilities that make it easy to use. nX also provides utilities for document preparation, text manipulation, macro processing, and syntax checking.

nX provides several facilities for development team coordination and communication, including the Revision Control System (RCS), the *make* and *pmake* utilities, and a mail system. RCS supports the management of large development projects, in which many programmers are working and where multiple versions of programs must be maintained. RCS manages the process of revising text files such as source code and documentation. It maintains a complete revision history, automates storage and retrieval of multiple versions of a program, provides release and configuration control, and controls programmer access to source code. RCS is a second-generation source control system that provides a higher-level command set than earlier systems, such as SCCS, and better support for large-scale development projects. RCS provides management facilities, for example, for multiple, parallel lines of development, and greatly facilitates the consolidation of changes from several such development lines into one version. Additionally, RCS internally stores and organizes change history in such a way that it can rapidly retrieve the most recent versions of a program, even when there have been numerous revisions.

In any development project, changes to code have to be reflected in the executable object module. In large projects, it can be difficult to maintain manually an up-to-date executable that correctly reflects changes, since the effects of changes in the code are difficult to track. *Make*, a utility for maintaining consistency between program source and object modules, keeps track of the interdependencies between program elements so that when a change is made in one element, make determines which others must be reprocessed to reflect the changes. *Make* then automatically recompiles or otherwise reprocesses dependent elements to make sure they are updated in a consistent manner. Standard UNIX operating systems provide single-processor *make*. nX provides a multiprocessing version of *make*, called *pmake*. Where possible, *pmake* automatically spreads recompilations over multiple processors, resulting in recompilations taking a fraction of the time required by *make*.

THE Xtra TOOL SET

The Xtra tool set provides a powerful platform for debugging, analysis, and performance tuning of multiprocessing programs executing under pSOS^{+m} and nX on the TC2000 computer system. Xtra exploits the latest advances in user interface technology to help software developers understand the behavior of their programs.

Xtra supports multiprocessing, parallel processing, and function card clustering. To program a multiprocessor system effectively, a programmer must understand the interactions between the elements of the program and the multiple processors of the system. Xtra helps the programmer gain this understanding by presenting a graphic view of the dynamic interactions among multiple processes and processors. The Xtra user interface builds upon the X Window System to present this view in a manageable, easy-to-understand format.

Xtra consists of a set of software instruments that let the user observe and measure the behavior of a program. Xtra Release 2.0 provides the first two instruments: the TotalView debugger and the Gist performance analyzer. A series of additional instruments and added functionality will be available in future releases.

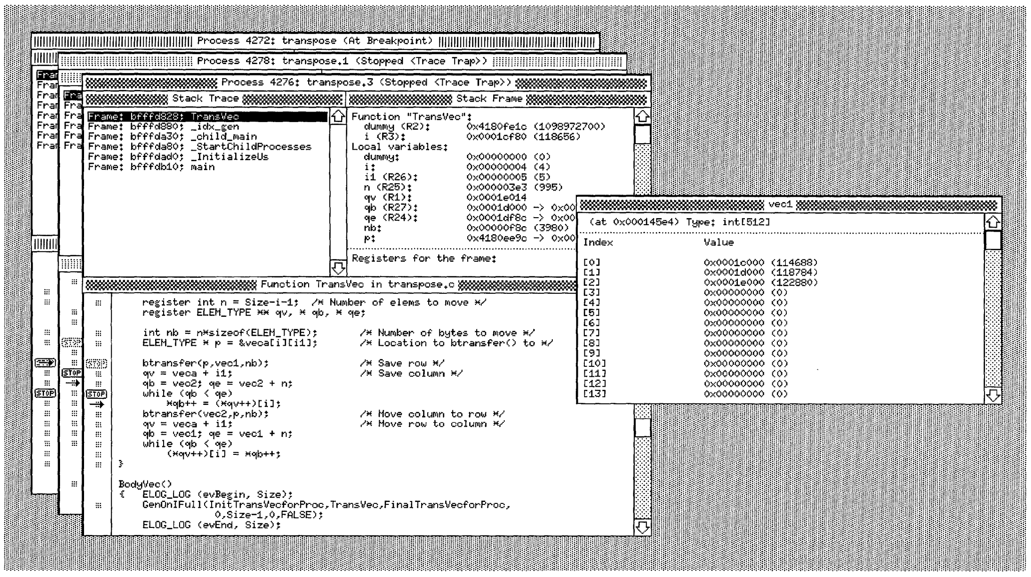
THE TOTALVIEW DEBUGGER

The TotalView debugger is the first commercial source-level debugger designed from the ground up to help programmers exploit the full power of an advanced multiprocessor. The TotalView debugger provides a simple but powerful means to observe, probe, and understand the complex, subtle, and sometimes surprising interactions inherent in a multiprocessor system.

Early multiprocessor debuggers merely extended conventional line-oriented debugger technology. The TotalView debugger builds upon the features of the X Window System. Conventional program debugging tools are line-oriented, presenting very little information at a time and generally concerning only a single aspect of program behavior. Users must enter dozens of commands, explicitly asking about each process by name and specifying which variables to print. The TotalView debugger's multiple-window display shows a full picture of the program's state automatically, helping the user grasp the nature of the problem at hand quickly .

The TotalView debugger's user interface provides a simple and intuitive way to interact with the program. Pop-up menu commands eliminate the need for command memorization, while keyboard equivalents for menu commands allow expert users to interact efficiently with the debugger. Mouse input provides a "point and click" mechanism for setting breakpoints.

An object-oriented design allows a user to "dive" into "objects," such as program variables, routines, or stack frames, to get progressively more detailed information on their structure or state. At any point, the user can call on the TotalView debugger's integrated, context-sensitive, on-line help facility. Help then automatically brings up information specific to the current debugging context.



```
Process 4272: transpose (At Breakpoint)
Process 4278: transpose.1 (Stopped <Trace Trap>)
Process 4276: transpose.3 (Stopped <Trace Trap>)

Stack Trace
Frame: bfffd828: TransVec
Frame: bfffd880: _idx_gen
Frame: bfffda30: _child_main
Frame: bfffda80: _StartChildProcesses
Frame: bfffdad0: _InitializeUs
Frame: bfffdb10: main

Stack Frame
Function "TransVec":
  dummy (R2):       0x4180fe1c (1098972700)
  i (R3):           0x0001cf80 (118656)
Local variables:
  dummy:            0x00000000 (0)
  i:                0x00000004 (4)
  i1 (R26):         0x00000005 (5)
  n (R25):          0x000003e3 (995)
  qv (R1):          0x0001e014
  qb (R27):         0x0001d000 -> 0x00
  qe (R24):         0x0001df8c -> 0x00
  nb:               0x00000f8c (3980)
  p:                0x4180ee9c -> 0x00
Registers for the frame:

Function TransVec in transpose.c
    register int n = Size-i-1;  /* Number of elems to move */
    register ELEM_TYPE ** qv, * qb, * qe;

    int nb = n*sizeof(ELEM_TYPE);        /* Number of bytes to move */
    ELEM_TYPE * p = &veca[i][i1];        /* Location to btransfer() to */

    btransfer(p,vec1,nb);                /* Save row */
    qv = veca + i1;                      /* Save column */
    qb = vec2; qe = vec2 + n;
    while (qb < qe)
        *qb++ = (*qv++)[i];
    btransfer(vec2,p,nb);                /* Move column to row */
    qv = veca + i1;                      /* Move row to column */
    qb = vec1; qe = vec1 + n;
    while (qb < qe)
        (*qv++)[i] = *qb++;
}

BodyVec()
{   ELOG_LOG (evBegin, Size);
    GenOnIFull(InitTransVecforProc,TransVec,FinalTransVecforProc,
               0,Size-1,0,FALSE);
    ELOG_LOG (evEnd, Size);

vec1
 (at 0x000145e4) Type: int[512]
Index               Value

[0]                 0x0001c000 (114688)
[1]                 0x0001d000 (118784)
[2]                 0x0001e000 (122880)
[3]                 0x00000000 (0)
[4]                 0x00000000 (0)
[5]                 0x00000000 (0)
[6]                 0x00000000 (0)
[7]                 0x00000000 (0)
[8]                 0x00000000 (0)
[9]                 0x00000000 (0)
[10]                0x00000000 (0)
[11]                0x00000000 (0)
[12]                0x00000000 (0)
[13]                0x00000000 (0)
```

The multiple windowing feature of the TotalView Debugger lets the user monitor and control several processes simultaneously. Panes within each window show the stack trace, stack frame, and executing code for each process. All windows relating to a particular process use the same title-bar fill pattern.

Debugging Multiple Processes with TotalView

When developers of parallel and multiprocess applications begin the debugging phase, they face the problem of understanding the effects of many processes running simultaneously. Processors, memory, and externally generated signals often interact in complex ways that are not manageable with conventional debugging tools. Even if users examine each process individually, the resultant picture is disjoint, and the effect of one process upon another is often lost.

To be truly useful for multiprocessor programs, a debugging tool must be able to detect events in one process and show the corresponding reaction of other processes. In addition, since so many interactions can take place at the same time, a multiprocessor-oriented debugger should automatically detect, acquire, sort, and present information that can give a user insight into the behavior of a program.

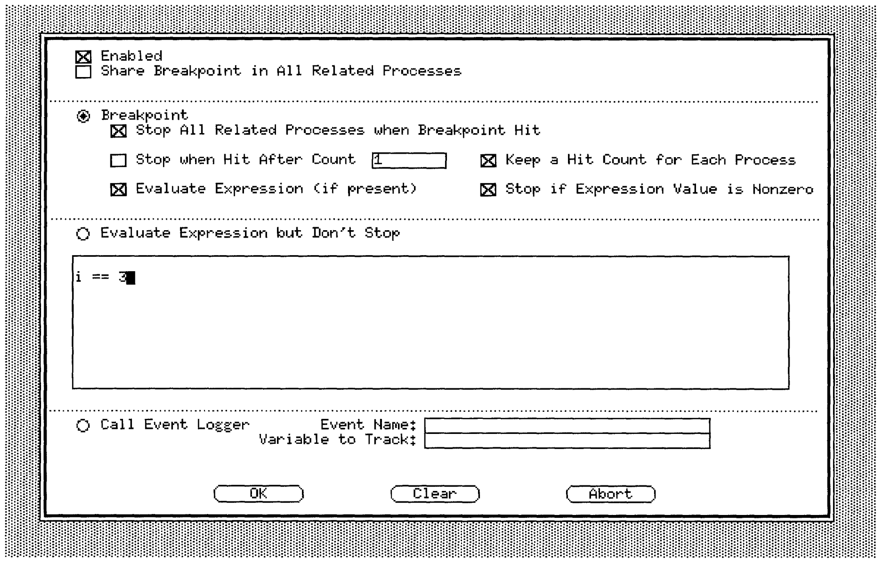
The TotalView debugger meets this challenge in a number of ways. It acquires and presents useful information in a user-friendly window-based display. Each executing process has its own window, and any or all windows may be open at a given time.

Automatic Process Acquisition

When an executing process spawns a child process (on the same or a different processor), the debugger automatically acquires control over that new process. The programmer does not have to direct the actions of the debugger. Process acquisition continues with all subsequently spawned processes, regardless of which process acted as parent. All processes run in parallel under the control of the debugger.

Automatic Display Updates

When a process running under TotalView stops, TotalView automatically updates all windows to display current values for the process's variables and state.



[✓] Enabled
[ ] Share Breakpoint in All Related Processes

(●) Breakpoint
[✓] Stop All Related Processes when Breakpoint Hit
[ ] Stop when Hit After Count 1 [✓] Keep a Hit Count for Each Process
[✓] Evaluate Expression (if present) [✓] Stop if Expression Value is Nonzero

( ) Evaluate Expression but Don't Stop

```
i == 3
```

( ) Call Event Logger Event Name:
Variable to Track:

"Diving" into a breakpoint causes this dialogue box to pop up.

Sharing Breakpoints

Users can readily set breakpoints in their programs by scrolling to the appropriate source-code line in the same pane and clicking the mouse on the shaded box to the left of the line. TotalView displays these boxes for all executable lines of code. In addition, users can easily modify the breakpoint control function through a dialog box (see figure above).

TotalView augments the usual breakpoint function for multiprocessing by letting a group of simultaneously-running processes share a single breakpoint. This allows users to examine specific processes either individually or as an interacting group. Programmers also have the choice of stopping any or all processes when they encounter an individual breakpoint. The ability to group processes and simultaneously stop them simplifies debugging multiprocessor applications.

TotalView can also set event points, which create event logs for Gist. See the Gist section for more information.

Observing Processes Not Started Under TotalView

In a multiprocessing environment, it is possible for unrelated programs to interact in unintended ways. Many debuggers can only monitor and control processes that start under the debugger. TotalView lets the user open a window on any process to observe its behavior or control it, regardless of how it was started. This enables users to debug a program that has hung during normal (non-debugger) execution.

The TotalView User Interface

TotalView offers a graphic interface with multiple windows, pop-up menus, and context-sensitive help. Users can enter commands using a mouse. For easy reading, all windows associated with a given process use the same graphic pattern in the title bar. TotalView graphic interface features include:

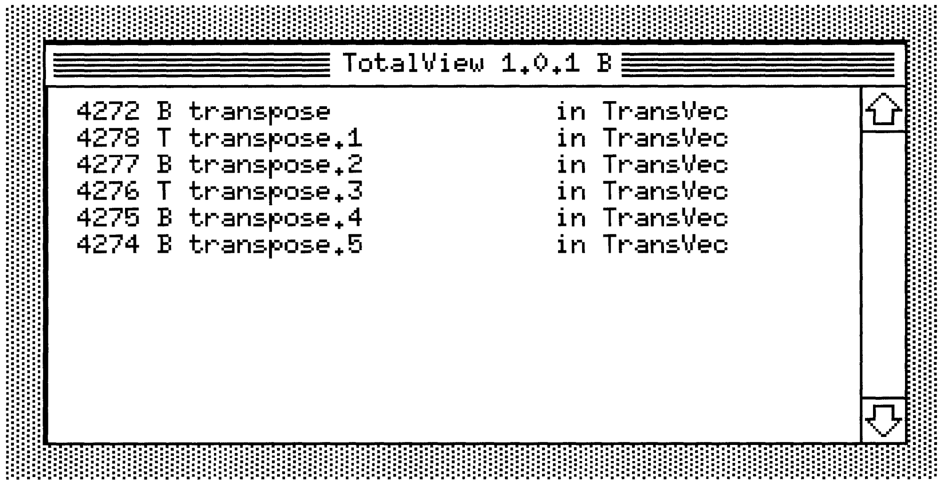
- A point-and-click style for selecting items of interest and setting breakpoints
- Pop-up menus with keyboard equivalents for the most common commands
- A window for each process that can be selectively displayed
- Windows for specific variables of interest, which are automatically refreshed when the corresponding running process encounters a breakpoint
- Context-sensitive help information
- Spelling correction for the names of routines and variables entered from the keyboard
- The ability to dive into objects such as routines, variables, data structures, and stack frames for a more detailed view

The Process Window

TotalView displays information about each process in a separate window. Each process window displays the process's stack trace, currently selected stack frame, and source code, using a separate pane for each. The window shows:

- The name, process ID, program counter and status of the process
- The stack trace and stack frame for the process, highlighting the current routine
- The source code (with assembly language display as an option) for the currently executing routine
- Any breakpoints and event points set in the source code

The process window can also be used to inspect core dump files. This helps the user navigate easily through the huge volume of data contained in typical core dumps.



```
4272 B transpose            in TransVec
4278 T transpose.1          in TransVec
4277 B transpose.2          in TransVec
4276 T transpose.3          in TransVec
4275 B transpose.4          in TransVec
4274 B transpose.5          in TransVec
```

The root window shows which processes are currently running (by name), their process ID number, and their status (e.g., stopped). Diving into a process in this window opens a window for the process.

The Root Window

The root window opens when the user invokes the TotalView debugger. The root window lists the name, process ID, status, and current routine executing for each process being debugged. The TotalView debugger updates it as executing processes change their state, and whenever a new process is spawned.

Additional Windows

The following windows can be opened:

- The Process Groups Window, which displays information about all of the processes being debugged
- The Global Variable Window, which displays the name and value of all global variables used by the process of interest
- Variable Windows, which display the address, data type, and value of specified variables, registers, or values stored in a block of memory
- Expression Evaluation Window, which allows evaluation of code fragments in Fortran, C, or Assembler, in the context of a specified process
- Breakpoints Window, which displays the line number, routine name, and source-file name for all breakpoints and event points set in the current process
- Unattached Processes Window, which displays the name, process ID, and status of all unattached processes running with the user's ID
- Event Log Window, which displays a history of significant events for each process
- Help Window, which displays information about the user's current debugging context

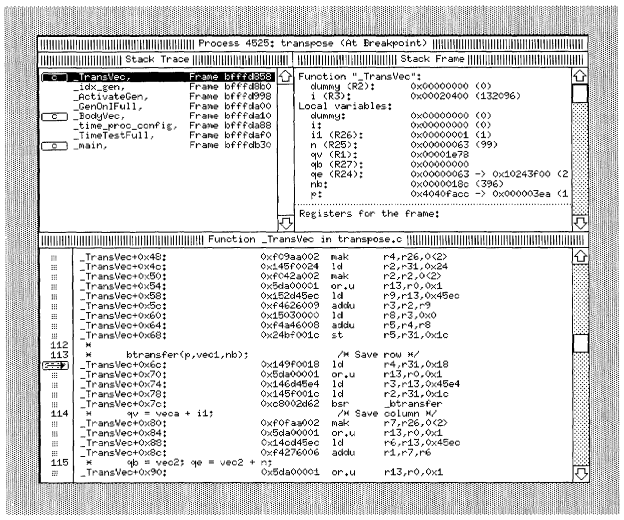
Object Management

The TotalView debugger treats processes, variables, routines, values, and stack frames as objects the user can manipulate. When a new process is created, the system refreshes the root window to reflect the processes' existence and status. The line describing the process is an object. The user can click on the object to dive into it, opening a complete process window for it.

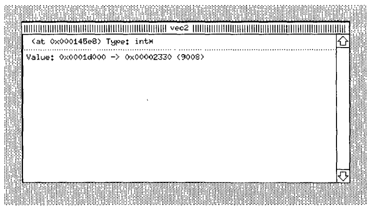
From a given process window, users can dive into other objects. Diving into a variable, for example, displays the address, data type, and value of the variable in a new window. Users can dive into a wide variety of objects, including processes, routines, variables, data structures, breakpoints, and stack frames, simply by pointing the mouse and clicking. Generally, a separate window appears to display the information, which users can click closed. In this way, users have complete control over the amount of information displayed on the screen.

THE GIST PERFORMANCE ANALYZER

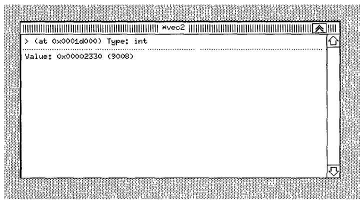
The Gist performance analyzer is a nonobtrusive, graphic tool for examining the dynamic behavior of multiple-process programs. Gist provides a graphical picture of the interaction of processes executing in parallel on a microsecond-by-microsecond basis. Gist displays supply the programmer with critical information for optimizing the performance of programs in a multiprocessor environment.



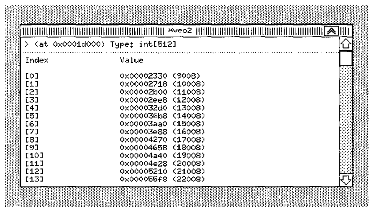
```
Process 4525: transpose (At Breakpoint)
Stack Trace                                   Stack Frame
 c  _TransVec,          Frame bfffd858   Function "_TransVec":
    _idx_gen,           Frame bfffd8b0     dummy (R2):      0x00000000 (0)
    _ActivateGen,       Frame bfffd998     i (R3):          0x00020400 (132096)
    _GenOnIFull,        Frame bfffda00   Local variables:
 c  _BodyVec,           Frame bfffda10     dummy:           0x00000000 (0)
    _time_proc_config,  Frame bfffda88     i:               0x00000000 (0)
    _TimeTestFull,      Frame bfffdaf0     i1 (R26):        0x00000001 (1)
 c  _main,              Frame bfffdb30     n (R25):         0x00000063 (99)
                                           qv (R1):         0x00001e78
                                           qb (R27):        0x00000000
                                           qe (R24):        0x00000063 -> 0x10243f00 (2
                                           nb:              0x0000018c (396)
                                           p:               0x4040facc -> 0x000003ea (1
                                         Registers for the frame:

Function _TransVec in transpose.c
 :::  _TransVec+0x48:              0xf09aa002  mak     r4,r26,0<2>
 :::  _TransVec+0x4c:              0x145f0024  ld      r2,r31,0x24
 :::  _TransVec+0x50:              0xf042a002  mak     r2,r2,0<2>
 :::  _TransVec+0x54:              0x5da00001  or.u    r13,r0,0x1
 :::  _TransVec+0x58:              0x152d45ec  ld      r9,r13,0x45ec
 :::  _TransVec+0x5c:              0xf4626009  addu    r3,r2,r9
 :::  _TransVec+0x60:              0x15030000  ld      r8,r3,0x0
 :::  _TransVec+0x64:              0xf4a46008  addu    r5,r4,r8
 :::  _TransVec+0x68:              0x24bf001c  st      r5,r31,0x1c
 112   *
 113   *      btransfer(p,vec1,nb);             /* Save row */
 ==>  _TransVec+0x6c:              0x149f0018  ld      r4,r31,0x18
 :::  _TransVec+0x70:              0x5da00001  or.u    r13,r0,0x1
 :::  _TransVec+0x74:              0x146d45e4  ld      r3,r13,0x45e4
 :::  _TransVec+0x78:              0x145f001c  ld      r2,r31,0x1c
 :::  _TransVec+0x7c:              0xc8002d62  bsr     _btransfer
 114   *      qv = veca + i1;                   /* Save column */
 :::  _TransVec+0x80:              0xf0faa002  mak     r7,r26,0<2>
 :::  _TransVec+0x84:              0x5da00001  or.u    r13,r0,0x1
 :::  _TransVec+0x88:              0x14cd45ec  ld      r6,r13,0x45ec
 :::  _TransVec+0x8c:              0xf4276006  addu    r1,r7,r6
 115   *      qb = vec2; qe = vec2 + n;
 :::  _TransVec+0x90:              0x5da00001  or.u    r13,r0,0x1
```



```
vec2
 (at 0x000145e8) Type: int*
Value: 0x0001d000 -> 0x00002330 (9008)
```



```
*vec2
> (at 0x0001d000) Type: int
Value: 0x00002330 (9008)
```



```
*vec2
> (at 0x0001d000) Type: int[512]
Index               Value

[0]                 0x00002330 (9008)
[1]                 0x00002718 (10008)
[2]                 0x00002b00 (11008)
[3]                 0x00002ee8 (12008)
[4]                 0x000032d0 (13008)
[5]                 0x000036b8 (14008)
[6]                 0x00003aa0 (15008)
[7]                 0x00003e88 (16008)
[8]                 0x00004270 (17008)
[9]                 0x00004658 (18008)
[10]                0x00004a40 (19008)
[11]                0x00004e28 (20008)
[12]                0x00005210 (21008)
[13]                0x000055f8 (22008)
```

Double-clicking on a variable in the source code pane opens a second window with further information about that variable. In this example, since the variable is a pointer to another variable, diving on the pointer in turn opens a third window. The fourth window in this figure shows the results of screen editing the type of variable. Clicking on the chevron symbol returns to the previous window. Arrows in the left-hand column indicate the program line at which the process is currently stopped.

The Gist performance analyzer consists of two parts: a library of event-logging functions and the Gist display program. Programmers use the event-logging functions to instrument programs so that the program generates an event log file when executed. The Gist program graphically displays the event log and provides an array of analytical functions.

Gist is integrated with TotalView. A programmer can set Gist event points to instrument a program just by clicking at lines in the source pane. Once the Gist data has been collected with TotalView, it can be reviewed with the TotalView Gist command. Gist can also be used as a stand-alone tool without TotalView. In this mode of operation, the programmer places Gist event logging calls directly in the program source code.

With Gist, a programmer can quickly identify areas worthy of performance tuning. For example:

- Gist can identify performance bottlenecks. Some multiprocess programs have clear synchronization points (barriers) that all processes must reach before the program moves to the next stage of processing. A single lagging task in an earlier stage can cause all subsequent tasks to be delayed. Gist helps programmers find such bottlenecks quickly and easily, so that lagging tasks can be eliminated.

- Gist can examine and quantify program dynamics. Programmers can use Gist to focus on areas of the program critically important to the performance goals of the application. Programmers can characterize the variability of performance of a particular task, which was caused by such external factors as operating system overhead.

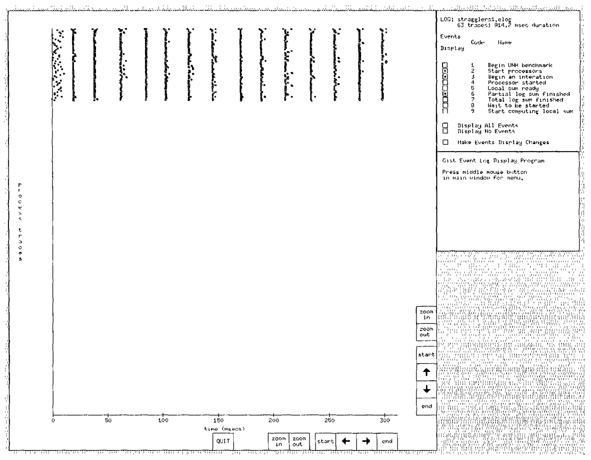
- Gist can examine resource usage. As a graphical profiler, Gist can identify the tasks and routines that consume the most computation time. The programmer can then choose to optimize the routines that have the maximum impact on overall performance.

As with TotalView, the Gist interface is based on the X Window System. When programmers use it with TotalView, they invoke the Gist command from a process window. This opens a Gist window, displaying the most recently created event log for the program. Gist provides a pop-up command menu and dialog boxes to specify commands along with detail boxes that contain more information about a given event or state. Through the window manager, programmers can resize Gist windows or change the font size. Users can also specify where Gist events should be logged by clicking the mouse in the source pane, as if to set a breakpoint.

A Gist event-log file records when processes reach specific points, called event points, in their code. Each event point records a time stamp, event code, and a user-specified data item. A programmer first decides the events to log, then instruments the program at the corresponding points to call event log library functions. Alternatively the programmer can use TotalView to set event points.

When logging an event, the programmer can select an interesting piece of data and record it with the event. For example, in a program that transposes a matrix, the programmer can log the matrix element being transposed. This data can be presented later as part of the event display.

After generating an event log, the programmer can run Gist to generate a time-based display of events.



```
LOG: stragglers1.elog
     63 traces; 814.7 msec duration

Events
          Code    Name
Display

[ ]        1     Begin UNH benchmark
[●]        2     Start processors
[●]        3     Begin an interation
[ ]        4     Processor started
[ ]        5     Local sum ready
[●]        6     Partial log sum finished
[ ]        7     Total log sum finished
[ ]        8     Wait to be started
[ ]        9     Start computing local sum

[ ]   Display All Events
[ ]   Display No Events

[ ]   Make Events Display Changes

Gist Event Log Display Program

Press middle mouse button
in main window for menu.
```

By zooming out, the programmer sees a larger view of the program. The program contains barrier synchronization points (vertical lines), where each process waits until all processes reach the barrier before moving to the next stage of computation. Note the presence of stragglers (dots).

Gist lets the programmer:

- Scroll through the event log
- Zoom in or out to show a smaller or larger portion of the event log
- Measure the time between events
- Change the scale of the time axis
- Search for a specific event
- Control which events are displayed
- Display detail boxes, which provide additional information about an event
- Write an event log to a text file for further processing by such programs as awk, sort, or sed
- Dump a screen to a file in PostScript format

As an alternative to the event display, Gist lets a programmer display a program's execution in terms of states. A state represents the period of time between two events. To define a state, the programmer specifies an entry event and an exit event. For example, a programmer could define states to show the following:

- Contention for locks. In a program that uses locks, a programmer could define a state that spans the period from when the process tries to get the lock to when it actually gets the lock

- Synchronization bottlenecks. In a program where processes wait for a synchronization event to occur, a programmer could define a state that spans the period over which a process waits.

Programmers can use Gist to display only events, only states, or both events and states. Since each state has a unique fill pattern, programmers can easily distinguish one state from another. Further, programmers can delete states, display detail boxes about each state, and write state definitions to a file.

For analyzing states, Gist provides the following tools:

- Histograms, which show, for a specific state, durations of each state occurrence. The histogram display also shows the average duration, maximum and minimum durations, and the standard deviation of the duration.
- State Durations, which show the percentage of time that each process, a single process, or all processes spend in each state

TC2000 COMPILERS AND LANGUAGES

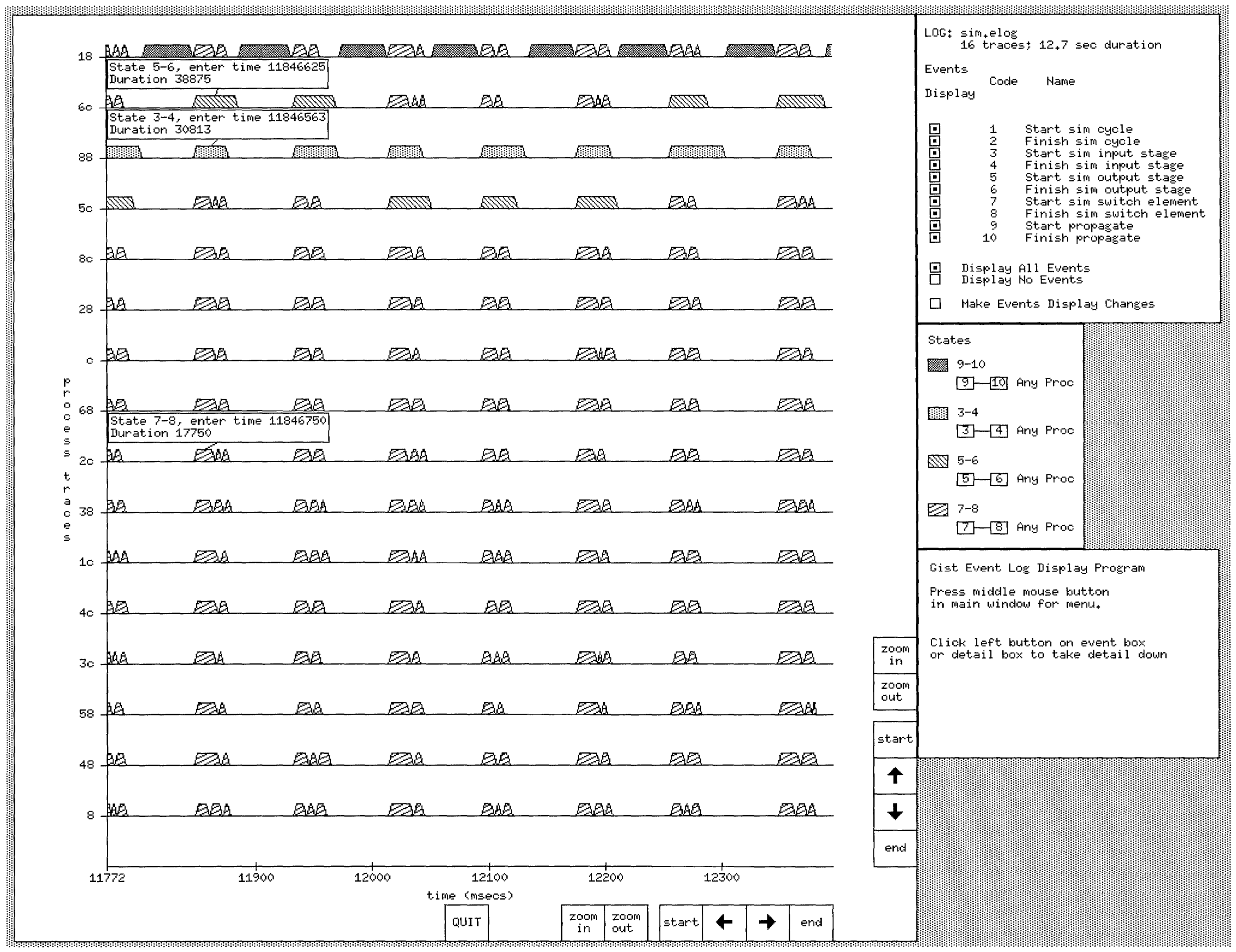
The TC2000 system provides the languages and compilers required for today's time-critical applications. First, for time-critical and scientific applications where Fortran programs are currently being used, TC2000 Fortran provides a familiar environment that lets the programmer adapt existing software to take advantage of parallel processing on the TC2000 multiprocessor system. Second, TC2000 Ada targets government and military applications, as well as projects in other areas that have the goal of developing readily documented, reusable software. Third, for systems programming, TC2000 C provides parallel programming facilities similar to those of TC2000 Fortran, and in fact shares the same back-end optimizer and code generator with that compiler. Lastly, for computations with the highest performance requirements, the TC2000 system provides an assembler.

TC2000 compilers work together with the TC2000 system to provide a seamless programming environment. All the compilers permit calls on routines written in the other languages and in assembler. Programs written in any language can make operating system calls directly to nX or to pSOS^{+m}. This means they can access all of the TC2000 system functions from either operating system.

All TC2000 compilers are highly optimizing, resulting in extremely fast, compact code.

TC2000 FORTRAN

TC2000 Fortran lets programmers develop and port code from other systems quickly and easily to the TC2000 machine. It adheres to both ANSI X3.9 1978 (Fortran77) and DOD MIL-STD-1753, and supports most VAX/VMS extensions and Berkeley 4.2 BSD f77 enhancements.



State 5-6, enter time 11846625
Duration 38875

State 3-4, enter time 11846563
Duration 30813

State 7-8, enter time 11846750
Duration 17750

```
LOG: sim.elog
   16 traces; 12.7 sec duration
```

Events
Display

Display	Code	Name
▣	1	Start sim cycle
▣	2	Finish sim cycle
▣	3	Start sim input stage
▣	4	Finish sim input stage
▣	5	Start sim output stage
▣	6	Finish sim output stage
▣	7	Start sim switch element
▣	8	Finish sim switch element
▣	9	Start propagate
▣	10	Finish propagate

▣ Display All Events
□ Display No Events

□ Make Events Display Changes

States

▩ 9-10
9—10 Any Proc

▦ 3-4
3—4 Any Proc

▧ 5-6
5—6 Any Proc

▨ 7-8
7—8 Any Proc

Gist Event Log Display Program

Press middle mouse button
in main window for menu.

Click left button on event box
or detail box to take detail down

A Gist State Display of a simulation program. The display shows that the computation represented by State 9-10 creates a performance bottleneck. This bottleneck is caused by serialized code; the programmer could improve performance significantly by parallelizing this section of the program.

Optimizations

The TC2000 Fortran compiler offers an extensive set of optimizations, including:

- Register optimization
- Subroutine and function-call optimizations
- Loop optimizations
- Code simplification
- Folding of constants
- Subroutine inlining
- Instruction scheduling

Register optimizations exploit the advantages of the fast register access in a RISC processor. The TC2000 Fortran compiler analyzes patterns of program flow and data access to ensure optimal use of registers. In particular, it uses registers to hold the most frequently used static addresses, local variables for subroutines and functions, and the most commonly used parameter values (using a technique called *allocation by coloring*), when appropriate. Through *register coalescing*, the compiler organizes the computation of expressions to ensure that values end up in the registers where they will be needed.

Subroutine and function call optimizations eliminate unneeded frame pointers, reduce register context, and eliminate entry and exit code for very short routines. Most parameters are passed to a subroutine or function call through registers rather than by placement on the stack. Inlining is also used to improve the efficiency of subroutine calls by actually copying the subroutine code and inserting it "in-line" with the main program at the calling line.

Since loops account for the greatest percentage of time in most programs, loop optimization is one of the most valuable ways to speed program execution. Among the methods the TC2000 Fortran compiler uses are:

- Strength reduction, which substitutes the addition of a constant to a register value for the multiplication of a loop index by a loop-invariant value, greatly simplifying the computation of subscripts in multi-dimensional arrays
- Compile-time computation of trip count, which replaces computed loop control variables with simpler expressions wherever appropriate
- Loop rotation, which moves termination tests to the bottom of the loop whenever possible, to reduce branching
- Loop invariant analysis, which removes calculations and expressions that do not change from the body of the loop
- Loop unrolling, which decreases the number of required loop iterations (as specified by the index number), by duplicating the loop contents two or more times within the loop.

Code simplification eliminates superfluous operations such as intermediate assignments to unnecessary local variables, unnecessary calculations in compound logical expressions, repeated calculations of common subexpressions, and unreachable code. Where appropriate, the compiler replaces instructions with faster executing equivalents. In scaling operations, for example, shifts replace multiplications.

The TC2000 Fortran compiler further optimizes expressions by folding constants. These compile-time optimizations include expression evaluation, expression simplification, and type conversion in mixed-mode expressions.

When requested by the user, TC2000 Fortran can perform inlining of subroutines. Inlining substitutes a called subroutine's code directly into the calling program, rather than branching to common subroutine code. This improves performance in two ways. First, the inlining eliminates the overhead of calling the subroutine. Second, the compiler can perform other optimizations on the inline code, tailoring the code to the specific use.

To branch to a subroutine with arguments, the compiler must place arguments in registers or on the stack, as defined by the 88000 calling conventions, and produce a branch instruction. To return, the subroutine must execute an instruction and pop the stack. With the subroutine code inline, the compiler can eliminate the branch and return instructions, as well as argument setup and cleanup. Since the inline subroutine code can use arguments where they lie, rather than requiring them to be in specific registers, the compiler can perform more effective register optimizations.

To maximize the utilization of the pipelines and multiple execution units in the Motorola 88100 processor, TC2000 Fortran adjusts the order of instructions while taking care to preserve correct program execution. The resulting instruction stream takes maximum advantage of the parallelism built into the 88100 processor.

Fortran Extensions

Some of the extensions beyond Fortran 77 that are included in TC2000 Fortran are:

- Additional data types:
  INTEGER*1, INTEGER*2, INTEGER*4
  LOGICAL*1, LOGICAL*2, LOGICAL*4
  REAL*4, REAL*8
  COMPLEX*8, COMPLEX*16, and DOUBLE COMPLEX
- Additional statements:
  END DO
  DO WHILE
  INCLUDE
  IMPLICIT
  READ and WRITE past END-OF-FILE
- Bit operations:
  Bit Field Manipulations
  Bit Subfields
  Bit processing
  Bit Constants
- Dynamic Allocation of Variables (for programs running under the nX operating system)
- Interlanguage Procedure Calls to C and Assembly Language Routines

Parallel Programming Extensions

The TC2000 Fortran compiler extends the basic Fortran language definition to provide a powerful set of parallel-processing capabilities for programs running under nX. These extensions support a highly flexible parallel-processing model useful in a wide range of applications. They include the ability to share variables and common blocks, to interleave arrays across processor memories to increase effective memory bandwidth, and to specify parallel execution of code fragments.

TC2000 Fortran gives the programmer a choice of two different methods for invoking parallel execution: language extensions and compiler directives. The first involves using a simple set of extensions to the basic Fortran statements. These extensions capture the essence of standard Fortran syntax and reflect the standards in parallel programming languages being pioneered by the Parallel Computing Forum. The result is a programming language that can exploit the power of large-scale parallel processing, while simultaneously making experienced Fortran programmers feel comfortable.

The extended commands for parallel processing include:

- PARALLEL DO, which specifies that the iterations of a DO loop are to be executed in parallel. PARALLEL DO includes options that give the user control over the granularity of the parallel units and let the user control the number of levels of loop nesting to be parallelized
- The PARALLEL REGION command, which specifies that a block of code should execute in parallel on all available processors in a cluster
- The PARALLEL SECTIONS command, which, when used with the SECTION definition command, specifies that a set of sub-blocks should execute in parallel

Alternatively, the programmer can invoke these extensions using compiler directives. Programmers can use compiler directives to maintain a single set of source code for compilation on different computers. TC2000 Fortran directives appear as comment statements when read by other compilers.

TC2000 ADA

TC2000 Ada is a port of the TeleSoft TeleGen2 Ada development system to the TC2000 system. Providing a complete Ada development and execution environment under nX, along with an execution environment for pSOS^{+m}, TC2000 Ada complies fully with the ANSI/MIL-STD-1815A specifications and will be validated by the Ada Joint Program Office.

The system includes a full Ada Programming Support Environment (APSE): second-generation TeleGen2 Optimizing Ada Compiler, Library Manager and Library Toolset, Global Optimizer, Ada Profiler, Source Level Debugger, and Language Tools. On the TC2000 system, it lets the user make full use of both a powerful multi-processor and a modern language designed for code modularity and reusability.

TC2000 Ada supports large-scale, mission-critical software development projects. The compiler generates exceptionally fast, production-quality code. When used with the optionally invoked Global Optimizer, it can make compiled programs even more efficient. The compiler uses advanced error recovery techniques to speed up the development process. The Source Level Debugger lets a program be examined intermittently at the Ada source code level while the program is executing. The Language Tools help keep track of Ada packages and their contents, maintaining them in a consistent format within a programming project. The Library Manager and Library Toolset support large, complex programs and large programming teams by enabling versatile configuration management as well as library access and manipulation. The compiler and toolset are written in Ada and are self-compiled, demonstrating the toolset's usefulness for applications comprising over 450,000 lines of code.

TC2000 Ada programs have complete access to all features of the TC2000 shared memory, multiprocessor architecture. All TC2000 system functions are accessible from Ada. This enables Ada programs to invoke the standard UNIX and pSOS^{+m} system services as well as all the multiprocessor enhancements of nX and pSOS^{+m}. For example, an Ada program can create processes on specific processors, establishing shared memory between them, and use a broad range of disciplines for interprocess interaction, ranging from shared memory to message passing, to interacting with its own processes and the processes of other programs.

In the initial TC2000 Ada release, all Ada tasks in an Ada program execute in a single nX process. In short-term releases, the Ada rendezvous mechanism will be usable within an nX process or pSOS^{+m} processor for synchronizing tasks. In long-term TC2000 Ada releases, the Ada runtime system will be enhanced to schedule the tasks of an Ada program across multiple nX processes, or multiple pSOS^{+m} processors, within a cluster.

Ada Programming Support Environment (APSE)

TeleSoft's second-generation Optimizing Ada Compiler generates fast and efficient production-quality machine code. Engineers can use standard nX editors and utilities to maintain the Ada source text. An on-line help library contains documentation of all TC2000 Ada components and their usage. The combination of the TeleGen2 Ada Development System and nX tools provides software engineers with a full Ada Programming Support Environment (APSE).

Optimizing Compiler Structure

The TC2000 Ada compiler consists of four main components: the Front End, the Middle Pass, the Global Optimizer, and the Optimizing Code Generator. The Front End accepts Ada source text as input, performs lexical syntactic and semantic analysis, and produces a compact, intermediate language representation known as High Form. High Form is similar to Diana, but it requires less than one-tenth the space that Diana requires for typical expressions.

The Front End can produce a diagnostic error listing which intersperses error messages with the Ada source statements. The Front End employs innovative syntactic and semantic error recovery techniques to facilitate program development. These special "auto-correcting" techniques minimize spurious error messages and allow the detection of as many errors as possible during a single pass of the compiler. To help the programmer pinpoint the problem, most error messages cite the relevant section of the Ada Language Reference Manual (LRM), right down to the paragraph.

The Middle Pass reduces High Form to a lower semantic-level intermediate representation known as Low Form. Low Form lends itself to optimization and efficient code generation.

The compiler provides three levels of optimization. Ada engineers can develop code using the lowest level (Level 0) for compiler efficiency while reserving use of the more powerful levels (Level 1 and Level 2) for the highest code quality as projects near completion. More specifically, the three optimization levels perform the following functions:

- Level 0 includes straightforward Middle Pass processing with conservative setting of informational attributes. Aimed toward robustness and the ability to debug code, this level performs no optimizations that change the structure or operation of user code. Level 0 does not run Global Optimization.
- Level 1 produces efficient code while still achieving good compilation rates. It includes extensive Middle Pass processing for attribute information as well as optimization phases that do not require data flow analysis.
- Level 2 includes data flow analysis and related functions. It produces full production-quality code, including graph-coloring-based register allocation and interprocedural analysis.

The Global Optimizer, which is invoked at Level 1 and Level 2, takes Low Form as input and creates a faster and more compact version of the Low Form representation. The Optimizing Code Generator translates Low Form into efficient machine code, performing additional optimizations specific to the 88000 in the process.

The many high-performance optimization techniques performed by the TeleGen2 Optimizing Compiler include:

- Graph-coloring-based register allocation
- Arithmetic strength reduction
- Strength reduction on loop induction variables
- Full data flow analysis
- Check removal
- Loop invariant code motion
- Optimal subprogram ordering
- Parameter reordering
- Boolean expression reduction
- Common subexpression recognition
- Subprogram inlining
- Lifetime minimization
- Dead code elimination
- Instruction selection
- MC88000 instruction scheduling

Library Manager and Library Tools

TC2000 Ada provides a powerful Library Manager and Library Toolset suitable for large-scale development projects. The library management facilities provide fast access to previously compiled Ada units. They also efficiently handle multiple compilation units within the same file as well as lists of units in separate files submitted to the compiler.

The Library Management Toolset supports:

- Separate compilation
- Version control/configuration management
- Order of compilation rules
- Elaboration ordering
- System builds

The Library Manager and Toolset offers an easy-to-use and flexible user interface with many options for manipulation and displaying Ada libraries and sublibraries. A sublibrary is the basic portion of the library that contains compiler-produced code and information. These sublibraries facilitate large development projects by providing:

- The use of multiple directories
- Read-write and read-only sublibraries, designed for large projects where engineers have private experimental versions and shared stable versions
- Shared access or private access to sublibraries
- Handling of virtually unlimited numbers of compilation units

The Library Toolset provides the ability to:

- Create, copy, or delete Ada sublibraries
- Copy, move, or delete compilation units from or between sublibraries
- View the contents of a library
- Prepare lists of various compilation unit dependencies in different formats
- Prepare recompilation reports, from which compiler command files can be generated

Developers can easily use the Library Manager and Toolset from an interactive terminal. These features perform lengthy or repetitious library management tasks conveniently through the use of command files, resulting in a minimization of recompilation overhead.

Execution Environment

TC2000 Ada includes two complete Ada Execution Environments (AEE) which provide all the necessary support routines for executing an Ada program under nX or pSOS$^{+m.}$ These routines include Run-Time Support Packages, Target-Dependent Packages, Code Generator Support, and File I/O Support.

The AEE also includes several important pre-defined packages, such as the standard Ada packages:

- TEXT_IO

- SEQUENTIAL_IO
- DIRECT_IO
- IO_EXCEPTIONS
- UNCHECKED_CONVERSION
- CALENDAR

The compiler implements selected features of the Ada language by inserting calls to the Run-Time Support Package (RSP). The RSP performs tasking (including interrupt handling) and other run-time functions, such as providing values for 'IMAGE and 'VALUE attributes. The RSP is written almost entirely in Ada, with a small number of packages implemented in assembly language.

The AEE also contains Code Generator Support (CGS) routines which provide low level support for exception handling, dynamic memory allocation, and handling of composite types. The File I/O packages provide an interface to files and services used by the standard Ada I/O packages. Routines also support Ada developers in interfacing to nX or pSOS^{+m} system services and languages.

Profiler

The Ada Profiler lets the software engineer monitor, from a high level, the flow of a program and to tune the application from the resultant information provided by the tool. The Profiler is a program-execution monitoring tool which captures and displays the Ada program's run-time performance, enabling tracking of inefficiencies in large, complex Ada applications. The profiler works at the subprogram level; it uses Ada compilation unit and subprogram names to capture information on the call graph for who called whom, and how many times, as well as the timing data. The profiler runs report programs to generate the desired listing. Options to organize or reduce the listing information for ease of use are available.

Source Level Debugger

The Source Level Debugger provides an easy way to examine and understand the behavior of compiled Ada programs. The debugger lets developers interact with the compiled program as if the Ada source itself were executing. It uses Ada scoping rules and semantics, refers to variables by name, prints variables by type, and refers to locations by compilation unit name and line number. Experienced developers can use the debugger to work at the machine code level.

Using the Source Level Debugger, designers can establish an interactive debugging session with an executing program. This greatly facilitates the debugging process since debugging activities can be adjusted to cope with the behavior of an executing program. Log and script files can also automate repetitive debug operations.

The Global Optimizer

The Global Optimizer increases the speed and compactness of the code produced by the compiler and generates a diagnostic report to improve programmer productivity. The Global Optimizer makes Ada programs as efficient as possible while encouraging developers to use Ada-oriented methodologies. The optimizer recognizes constructs that optimize maintainability and ease of design, but may result in "non-optimal" code. The optimizer modifies the resultant code to be faster and smaller than code produced by conventional compiler technologies.

The Global Optimizer analyzes the use of code among a collection of packages rather than within a given package. For example, a program may have many small procedures in separately compiled packages for ease of maintenance. The optimizer determines which procedures are only called once and inserts them in-line in the code stream to save the overhead of a procedure call. The result is an increase in the speed of the target code and a decrease in its size.

Language Tools

The Host Development System includes a suite of language tools designed to help large development projects run more smoothly. The Language Tools include the following:

- Ada Cross Referencer
- Ada Source Dependency Lister
- Ada Source Formatter
- Compilation Order Tool

The Ada Cross Referencer prepares a list of all symbolic names referenced within a program, showing where they were declared and referenced. This listing is most useful to programmers during the debugging and program analysis phases of a project.

The Ada Source Dependency Lister (ASDL) produces a valid compilation order list from a program containing many source packages. The tool generates a report listing the imports and dependents of each package, indicating which units must be compiled before another specific unit can be compiled. The ASDL also helps developers trying to interface to, or compile, another engineer's code by reporting in which source file the developer has located each compilation and by reporting a valid compilation order based on the current state of the source.

The Ada Source Formatter, often called a "pretty printer," reformats Ada source text to make it easy to read and consistent in appearance. This is helpful when programmers with diverging coding styles are working on the same project, or when they have to deliver an application in source code. Several formats are available, including that of the Ada Language Reference Manual (ANSI/MIL-STD-1815A).

The Compilation Order Tool provides the following capabilities:

- Generates a valid compilation order from a set of source files provided by the user that includes dependent units in the library that are obsolete
- Generates a valid compilation order from a given unit name (usually the main program) in the library
- Creates an input list file containing a valid compilation order
- Creates an executable script to do the compilation on request

- Automatically initiates the compilation if desired
- Generates a report consisting of unit names in valid compilation order, associate source file names, statistics, and undefined units
- Answers "what-if" questions without having to compile

TC2000 C

TC2000 C performs the same optimizations for run-time speed and code compactness as TC2000 Fortran does. It, too, adheres to industry standards; TC2000 C provides a superset of Kernighan and Ritchie's 1978 language definition (The C Programming Language) and includes ANSI C and Berkeley enhancements, including structure assignments, the use of structures as arguments, and enumerated data types. Like TC2000 Fortran, TC2000 C offers an extensive set of optimizations, including:

- Register optimization
- Subroutine and function-call optimizations
- Loop optimizations and unrolling
- Code simplification
- Folding of constants
- Instruction scheduling

THE UNIFORM SYSTEM

The Uniform System is a powerful, complete runtime environment for parallel processing designed especially to work with TC2000 C under nX. The Uniform System provides the efficient, transparent management of two resources key to effective parallel processing: memory and processors. It automatically creates a large shared address space to provide a single view of all memory, while effectively using the machine's full bandwidth. It maintains high processor utilization — and therefore efficiency in program execution — by dynamically assigning work to processors. A second, and equally important, benefit of dynamic load-balancing is that programs can be written independently of system or cluster size; a program developed and debugged on small systems or clusters can run in larger clusters without modification, and programs built for larger systems can run unchanged on smaller systems if processors are removed for maintenance. The Uniform System allows the programmer to concentrate on program development rather than on resource management.

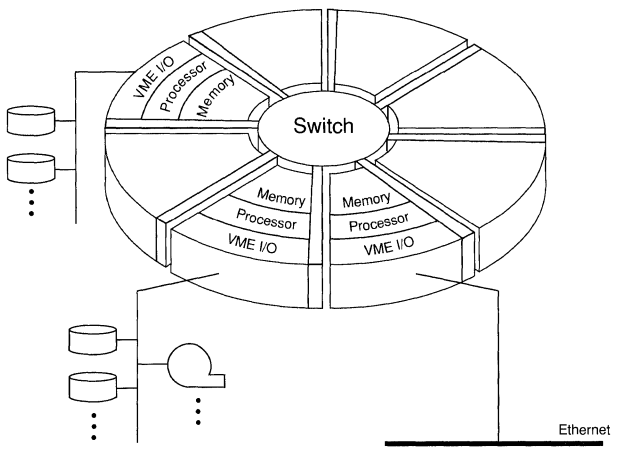
The Uniform System also provides the programmer with the explicit control required to fine-tune a program for optimal performance. The Uniform System provides utilities for explicit distribution and synchronization of data, task-to-processor assignments, and performance measurement. The combination of automatic resource management and explicit control gives the programmer the freedom to choose between rapid development prototyping and maximum performance.

When used in conjunction with clusters running pSOS^{+m}, the Uniform System can be used to bring the power of parallel processing to bear on time-critical applications. An example of this would be a high-resolution time-critical simulation in which the model is separable into two parts: a moving subsystem with behavior best described by a set of dynamic equations, and a rigid body characterized by a finite element model. The dynamic model would be best suited for execution in a time-critical cluster, running pSOS^{+m}, while the finite element analysis would be ideally matched to computation in a large parallel processing cluster using the Uniform System. The interaction between the moving and rigid subsystem models would take place through the use of common variables in shared memory.

I/O Subsystem and Peripherals

Each TC/FPV function card in a TC2000 system provides an independent industry-standard VME I/O interface. The VMEbus is the industry's most widely used 32-bit bus today. Since there are more than 2000 third-party boards available, users can have ready access to a host of third-party peripherals and boards, including A/D, D/A, parallel interfaces, and graphics processors. The interface conforms to IEEE 1014 and supports the following VME functional modules, as defined in the VMEbus Specification:[1]

- Master
- Slave
- Arbiter
- Requester
- Interrupt Handler
- System Clock Driver
- Bus Timer
- IACK Daisy-Chain Driver



Like processors and memory, the number of I/O channels in a TC2000 system is scalable. Every system has at least one VME I/O channel to support the standard peripherals, but can be configured with additional TC/FPV function cards to support applications with high aggregate I/O bandwidth requirements. Each interface can provide up to eight megabytes per second of VMEbus I/O throughput to separate, independent VMEbuses. The TC2000 system supports up to 320 independent VMEbus interfaces for a total aggregate I/O bandwidth of 2,560 megabytes per second. By providing scalable I/O, the TC2000 system maintains balanced performance capabilities across its full range and can support I/O-intensive applications. At its architectural limit of 504 I/O channels, the TC2000 system provides over four gigabytes per second of aggregate I/O bandwidth.

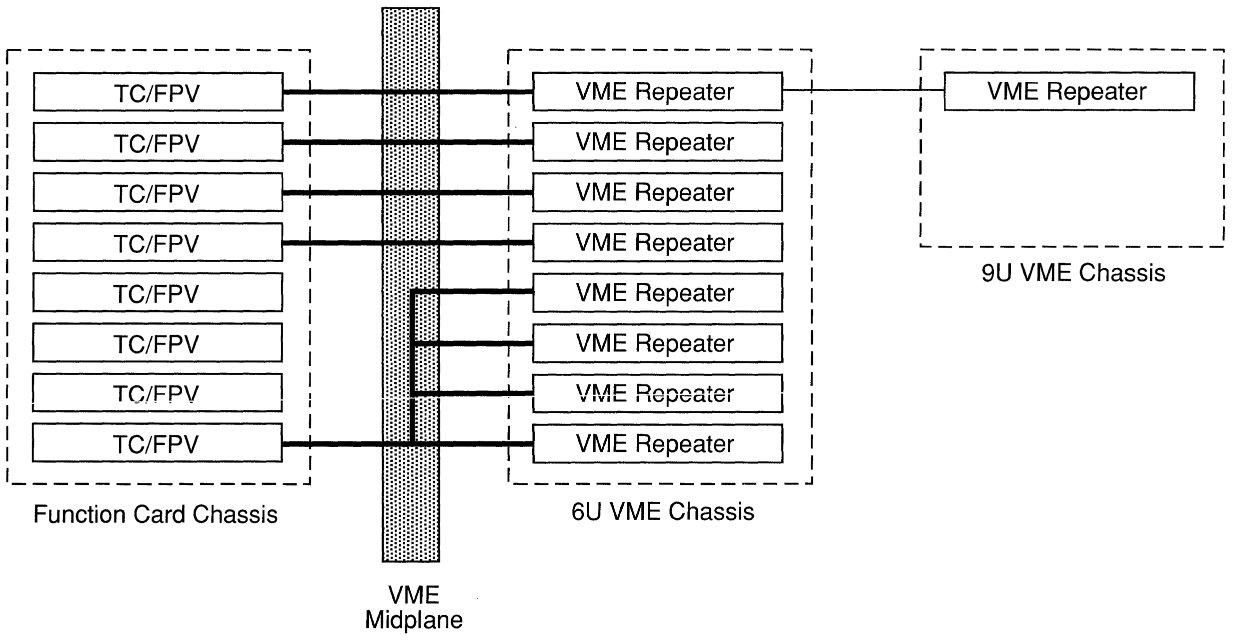
To make this scalable and flexible I/O subsystem easy to use, the TC2000 system integrates VME memory into the system memory address space. A 16-megabyte window into each VMEbus address space is mapped into the TC2000 system global address space, providing I/O resource access to both the local processor on the TC/FPV and to all other processors in the system. Similarly, devices on a VMEbus can access local memory on the attached TC/FPV and, through the Butterfly switch, can also access memory on other TC2000 function cards, letting these devices perform DMA transfers directly into any function card's memory.

The TC2000 machine uses the VME midplane to connect VME cards to function cards within a TC2000 expansion module. The VME midplane is passive like a standard VME backplane, but has function cards plugged into one side and VME cards plugged into the other side. The standard TC2000 VME midplane is internally wired to support five electrically separate and independent VMEbusses within a single chassis: four single-slot 6U VMEbusses and one four-slot 6U VMEbus. Each of the four single-slot VMEbusses connects to a different TC/FPV function card in the expansion module, providing a compact means to interface VME cards that require a dedicated channel to the TC2000 system's large number of VMEbus I/O channels. The four-slot VMEbus connects to a fifth TC/FPV in the expansion module,

[1]Motorola, Inc. The VMEbus Specification. Revision C.1, October 1985.

providing support for multi-board VME devices and for multiple devices. This configuration provides flexibility in integrating VME boards into a TC2000 system and avoids the expense associated with a large number of separate VME chassis and power supplies. Since it is passive, the VME midplane is relatively easy to customize for specific applications.

A high-performance, low-delay VME-to-VME repeater extends a TC/FPV channel to a standard 9U VME chassis. This lets users connect 9U VME cards or connect more than four cards to the same VMEbus channel.



VME cards connect to TC2000 function cards via the VME midplane. By plugging function cards into one side and standard 6U VME boards into the other side, the VME midplane hosts multiple VMEbusses within a single card cage.

The TC2000 system provides disk storage on high-performance 8-inch Winchester disk drives and 5 1/4-inch removable disk drives. The 8-inch disks support a peak transfer rate of 2.55 megabytes per second, with an average seek time of 17 ms and a latency of 8.2 ms. Using the scalable I/O capabilities of the TC2000 design, users can configure a system with multiple disk controllers. This kind of design will support applications with large disk space requirements or in applications, such as on-line transaction processing, that need many disk spindles to provide the required number of seek operations per second.

Each TC2000 system includes an industry-standard 1/4-inch cartridge tape drive, which is used for software distribution and for tape backup of systems with small amounts of disk storage. The drive uses the QIC-120 data format and records at a speed of 90 inches per second in streaming mode. Each cartridge holds up to 125 megabytes of data.

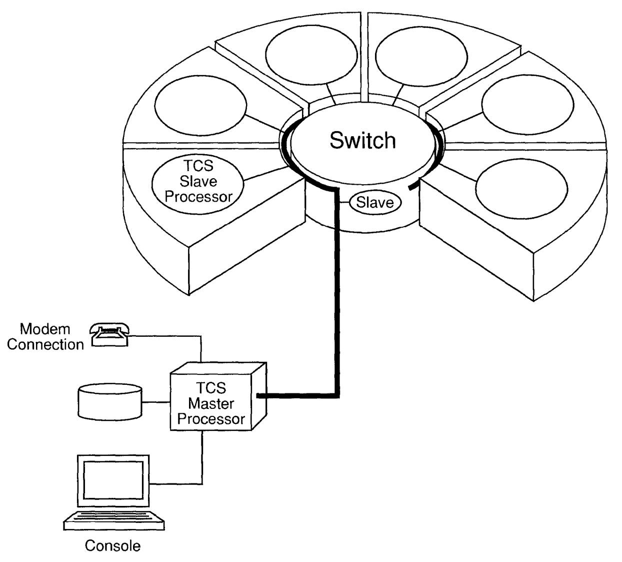
For backup of larger disk configurations and for media interchange with other computers, the TC2000 machine supports an optional 1/2-inch 9-track reel-to-reel streaming tape drive. This drive supports both ANSI X3.54 GCR (6250 bpi) and ANSI X3.39 PE (1600 bpi) data formats and operates in streaming mode at either 70 ips (GCR) or 100 ips (PE). Via a built-in cache, the tape drive is capable of start/stop emulation and provides a peak transfer rate of 632 KBytes per second. For ease of use, the drive provides automatic tape loading and threading.

The TC2000 system supports the IEEE 802.3 (Ethernet) networking standard. The hardware interface is fully supported by software in the nX operating system, providing TCP/IP networking protocols with up to four Ethernets per system.

The TC2000 architecture also supports asynchronous interfaces through standard RS-232C connections. The terminal controller and associated remote terminal cluster-controllers manage all serial line discipline issues, releasing the TC2000 system from high-interrupt service burdens. Each remote terminal cluster-controller supports up to 16 RS-232C asynchronous devices and connects, via coax, to the terminal controller. Users can daisy-chain multiple remote cluster controllers to provide support for additional asynchronous interfaces.

Availability, Reliability, and Maintainability

High availability requires a system to be highly reliable, to recover from failures quickly, and to be easy to repair. The TC2000 system provides high availability, reliability and maintainability through its support for a fully integrated maintenance and diagnostics subsystem and through BBN's manufacturing capabilities.



TEST AND CONTROL SYSTEM

The cornerstone of TC2000 maintainability is its Test and Control System (TCS). The TCS provides facilities for rapid fault detection and isolation, and controls the operation and configuration of the system to support rapid recovery and repair. The TCS consists of three components: 1) a master processor and associated software; 2) a slave processor for each function card and pair of switch modules in the system; and 3) a diagnostic bus that connects the slave processors with the master processor. The TC2000 system powers and clocks TCS components separately from the rest of the system.

The TCS master processor provides the user interface to the TCS; maintains the TCS software on its own disk drive, separate from the main system disk drives; and offers an optional removable disk drive for high-security applications. It coordinates the activities of the slave processors and handles messages received from the slave processors. The TCS is also responsible for loading bootstrap programs into function card memory at system boot time, which eliminates the need for bootstrap ROMs and simplifies the updating of bootstrap programs. Slave processors, which monitor various signals and sensors on each card, are implemented using a single-chip microprocessor.

Detection

The TCS slave processors are the key to rapid fault detection in a TC2000 system. Each slave continuously monitors its function card or switch module and reports any problems to the TCS master processor without impacting system performance. The slave processors monitor card temperature, voltage levels, and key signals such as the clock and the processor activity register (used to detect a stuck or hung processor). During the power-up sequence for a function card or switch module, the slave processor executes power-on tests to check out the card, such as memory and processor tests for a function card.

The TC2000 design supports preventive maintenance testing. The system runs on-line diagnostics under the nX operating system periodically to check the status of system components. Through the slave processors, the TCS can margin the voltage on cards up or down, enabling service personnel to detect marginal function cards prior to an actual failure.

Isolation

Once the system detects a fault, either by TCS monitoring or by the nX operating system, service technicians use TCS facilities to isolate the problem to a field-replaceable unit. The TCS master processor maintains a log on its disk of all problem reports from TCS slave processors and, when used with the nX console log also stored on disk, lets service people review the status of the system quickly. The TCS master processor supports a menu-driven user interface to a comprehensive suite of diagnostic programs and facilities for fault isolation. If the fault was severe enough to result in a crash of the nX operating system, service people can use *kdb*, the nX kernel debugger, to examine the cause of the crash in more detail.

The TCS master processor contains a modem and telephone line interface to help service people dial into the TCS and perform fault isolation from a remote location. They can remotely perform all TCS functions, including a review of the TCS and nX console logs. All function cards and switch module slave processors can report card type and revision levels, letting service people quickly display the configuration of a system. The dial-up interface to the TCS is password protected for security.

Recovery and Repair

If the problem resides on a function card, the TCS can logically disconnect that card from the system, enabling operation to resume after a failure. This disconnection occurs automatically for faulty function cards that are detected by system power-up tests, or can be performed manually via the TCS user interface. This ability to disconnect a function card also enables service personnel to run low-level diagnostics on that card while the rest of the system is operational. Stand-alone diagnostics can be loaded into the function card over the TCS bus and run under control of the slave processor.

To further reduce downtime and speed repair, TC2000 systems support power-on servicing of function cards. By distributing bulk power to the cards and converting this to the required voltages via DC-to-DC converters, the system lets most of the processors continue to run while service people remove the faulty function card and insert a new one. The next time the system is rebooted, it will automatically include the replaced function card into the system configuration. In summary, the mean time to repair (MTTR) for the TC2000 machine is extremely low, due to rhe modularity of the system design and the simultaneous power-on servicing features of the TCS.

RELIABILITY

BBN's central manufacturing organization, BBN Manufacturing Corp., serves all BBN subsidiaries and has over 10 years of experience in manufacturing reliable computer and electronics equipment. Reliability and quality are built in and checked at every step in the manufacturing process, from incoming parts inspection, through board and module test supported by automated test equipment, to extensive system-level testing. Every TC2000 system that leaves the production line undergoes a failure-free 72-hour burn-in in a hot/cold environmental chamber while running a suite of system-level exercise tests. Each system must also pass a detailed set of low-level diagnostic tests and a system-level stress test run under the nX operating system.

Glossary

APSE

Ada Programming Support Environment – the full set of software tools that are supplied along with a particular Ada compiler. This can include a global optimizer, a source-level debugger, a library manager, language tools, and a configuration-management toolset.

atomic operation

A set of operations that runs without interruptions from any other code on any processor in the system. Used, for example, to ensure that a process is not permitted to change a variable while another process is changing it.

bus-based connection

A traditional design for connecting multiple processors and memory units. All devices transfer information over a single data bus, and must share the bus bandwidth.

clusters

User-defined sets of processors functioning under the nX or pSOS^{+m} operating system. Clusters may be declared as private or shared.

communication latency

The length of time it takes for the transfer of information between one processor and non-local memory.

context switch

A context switch is a change in the thread of execution from one task to another. A task is, from the executive's perspective, a unit of execution which can compete on its own for system resources. A unit of execution is a set of operations that are carried out in series. Thus various tasks will be competing with each other for processor cycles in an application. The executive determines which task is to be running at any given time, and sets the processor context to that of the selected task. The change in processor context from one task to another is termed a context switch.

In nX, a context switch may involve swapping in and out of active memory of all information pertaining to the state of a process (i.e., the contents of its address space plus the contents of hardware registers and kernel data structures that relate to the process).

In pSOS^{+m}, a context switch can occur at system calls, after an interrupt is serviced, or in response to time passing if round-robin scheduling is enabled. When a task makes a system call that results in some other task with higher priority being made ready to run, pSOS^{+m} will context switch to the higher priority task. An interrupt handler can make system calls, but context switching will be deferred until the interrupt handler exits. Then if any of the system calls resulted in a task of higher priority than the interrupted task being made ready to run, pSOS^{+m} will context-switch to the higher priority task. Finally, if the application has enabled round-robin scheduling, when a task's timeslice has expired, pSOS^{+m} will context-switch another task at the same priority level that is ready to run.

crossbar connection

A design for connecting multiple processors and memory units with a separate physical connection between every processor and every memory unit. The complexity increases as the square of the number of connected units.

dbx

A standard line-oriented debugging facility within UNIX operating systems.

dedicated processors

A means whereby users can allocate one or more processors for their exclusive use. This TC2000 facility permits users to achieve run-times similar to those they would realize on a similar-sized machine running without any other users.

distributed computing

A computing environment in which two or more completely independent, self-contained computers are able to transfer data over a local or wide-area network.

execution environment

The term execution environment refers to the execution support provided to an application by the operating system or executive. This includes the facilities for task management, communications, synchronization, and memory management. The TC2000 system can support multiple execution environments with very different characteristics simultaneously on the same multiprocessor system. This is accomplished by allocating a cluster of processors that can be dedicated to a particular execution environment. The nX operating system is provided for timesharing operations such as those found in UNIX operating systems, with extensions to support TC2000 multiprocessing. The pSOS^{+m} executive is provided for real-time applications where the maximum in performance and predictability is required.

In addition to the general purpose nX and pSOS^{+m} execution environments, the TC2000 system provides Ada-specific execution environments for Ada applications. An Ada execution environment (AEE) is provided for both nX and pSOS^{+m}. The AEE provides support for Ada-specific concepts of tasking and memory management.

function card

The function card is the basic unit of functional capability (processing power, memory, and input/output busses) on the TC2000 system. Each function card is a self-contained Motorola 88000-based processor card. Function cards are connected together to share memory via the Butterfly switch.

Gist

The Gist performance analyzer is a graphical tool for displaying the dynamics of operation of multiprocessor applications on the TC2000 system. Gist can log events, and application states, with a clock resolution of one microsecond across the entire TC2000 system. The resulting data is conveniently presented in a graphical, interactive, X Window System based presentation. This allows the programmer to really view the dynamic interactions of various parts of the application and quickly get to the essence of any performance issues.

homogeneous

In multiprocessors, a design using functionally identical processing units.

hypercube

A design for connecting multiple processors where each processor is directly connected to "n" processors, and indirectly connected to all other processors. A processor can only reference its own physical memory; to access data in a different processor's memory unit, it sends an explicit message to software on another processor.

interrupt latency

External hardware devices usually request services of a processor by issuing an interrupt request. In the TC/FPV these requests come in over one of the seven VMEbus interrupt request lines. The interrupt request is a request that the processor stop executing the code thread it is in at the time of the interrupt and start processing the code associated with the interrupt handler of the device requesting the interrupt. *Interrupt latency* is the time from the interrupt request on the VMEbus, until the first line of code in the device interrupt handler is executed.

load-balancing

A method of spreading the total program workload over all the processors available in a multi-processing computer, with the goal of achieving a uniform processing load across all of the processors. This is easily and efficiently implemented on TC2000 computers with the Uniform System parallel-processing runtime environment.

local memory

Memory that is physically located on a function card, and therefore accessed by that processor somewhat faster than the memory physically associated with some other processor.

MIMD

Multiple Instruction Multiple Data. A method of parallel processing in which two or more processors can work on independent and different instructions simultaneously.

MIPS

Million Instructions Per Second. A unit of measure for the processing power of either an individual processor or an entire multiprocessing computer. Most people approximate the power of one VAX-11/780 computer as one MIPS. Used here as MIPS based on Dhrystones, that is, the performance on the Dhrystone 2.1 benchmark relative to the VAX -11/780.

multiprocessor

A computer that contains two or more central processing units (CPUs), working on one or more programs simultaneously, and often sharing memory, operating system kernels, and I/O management capabilities.

multistage switch connection

A design for connecting multiple processors and memory units with multiple paths, where the path used is determined at the time of sending the desired information. It is a serial decision network, and expands readily to accommodate hundreds of processors and memory units. Compared to the conventional crossbar, a large-scale multistage switch is simpler and more cost-effective.

nX

The operating system (based on the UNIX operating systems) for the TC2000 multiprocessing computer. It is POSIX-compatible with the UNIX 4.3BSD kernel, and also provides a rich and reliable multiprocessing environment. nX has an extremely flexible virtual-memory implementation, and supports networking protocols including TCP/IP. nX provides all the traditional advantages of the UNIX operating systems, such as time-sharing of resources and a wide range of excellent development tools.

paging

A memory management approach used when a processor does not have enough physical memory for all processes. The operating system kernel moves pages of memory between primary and secondary memory storage as required.

physical memory

The term physical memory is used to refer to memory that is physically present in the system — in contrast with virtual memory. An nX application's virtual memory may or may not be in physical memory at any given time. Some of the virtual memory may have been paged out to disk. Any reference to a paged-out area would have to wait until the page was read into physical memory. The advantage of virtual memory is that an application can have more virtual memory than the system has physical memory. pSOS^{+m} applications always execute out of, and hold data in, physical memory. This assures predictable performance for the application.

pipelining

Pipelining is a technique for improving the throughput of system by staging complex operations in a pipeline of processing elements. For example, a particular machine instruction might take five clock cycles to complete. If a single stage processing element handles this operation, then one operation can complete every five clock cycles. Consider a five-stage pipeline of processing elements, such that each stage of the pipeline can handle one clock cycle's worth of the instruction. A single instruction will still require five clock cycles to complete; however, as soon as it moves on to the second stage of the pipeline, another instruction can start in the first stage. Thus a new instruction can start every clock cycle, and as many as five instructions can be in progress at any given time. The five-stage pipeline has improved the processing throughput by a factor of five. Note that the latency of the operation has not been improved. Thus the key to effectively using a pipeline is to keep it as full as possible.

pSOS^{+m}

pSOS^{+m} is a light weight, real-time executive. pSOS^{+m} is designed to minimize the impact of the executive in real-time applications by minimizing scheduler overhead, context switch time, and interrupt latency due to critical sections of executive code where interrupts are disabled.

real-time

The term real-time is used to refer to applications which interact with, or simulate, the real-world in real-world time. Thus the timing of the application is precisely dictated by the physical properties of the world and the system being simulated or monitored. This timing is critical to get the correct answer. Since such applications demand real-world timing, performance must be guaranteed in an absolute sense rather than statistical average. For example an airline reservation system needs to achieve a certain average response time so that the customers will be happy - but it is acceptable that it sometimes take longer than this average to respond. However a flight training simulator must respond on time, every time, otherwise the simulator will not be correctly simulating the flight dynamics of the aircraft.

remote memory

Memory that is accessible to a given processor, but is physically located on some other function card. Ideally, access to remote memory is transparent to the user.

remote procedure call

Similar to a normal procedure call except that the body of the procedure is executed on some machine other than the TC2000 system, on a network of computational machines.

RISC

Reduced instruction set computer (RISC) refers to a computer that has been designed with the approach of minimizing the number and complexity of the instruction set. Generally the goal is to minimize computer hardware complexity, so that instructions can execute in minimum time, and hardware (chip) real-estate (that would have been used to implement more complex instructions) can be used for other functions to improve overall performance.

This approach contrasts with complex instruction set computers (CISC). These designs provide elaborate instructions (usually with the intent of minimizing the complexity of assembly-language programming and compiler design), by providing commonly used series of operations in a single instruction. Typically a RISC architecture will not provide the complex addressing modes and diverse source/destination options that CISC architectures do. For example, a typical CISC machine can move data from one location in memory to another in a single, if somewhat complex, instruction. RISC machines typically only offer "load and store register" instructions, so two RISC instructions will be required to accomplish the effect of the more complex "move memory to memory" instruction.

However, consider the complexity of the move memory to memory instruction. If it allows indirect addressing and requires more than a 32-bit word of memory for the instruction, as many as six pages of memory must be touched just to execute this one instruction (two for the instruction, one for each indirect address, one for the source and one for the destination). All of this requires a complex instruction decoding mechanism. In a RISC architecture, the chip real estate is used instead for pipeline, cache, and other mechanisms that improve the overall throughput of the system.

scalability

Scalability refers to the fundamental design concept of the TC2000 system — the ability to scale to hundreds of processors, memory modules and input/output busses. Scalability is achieved by placing function cards on the Butterfly switch. The Butterfly switch provides an interconnect mechanism that can economically grow with the number of function cards while avoiding access bottlenecks.

shared memory

A type of multi-processing memory architecture where data is stored in a globally-shared memory and can be accessed by all processors, analogous to posting a messsage on a bulletin board for anyone to read.

shell

Shell is a UNIX operating systems term for a utility program that interprets user commands. The shell is actually just an ordinary program that is automatically started for the user when he logs into the system. In addition to directly accepting typed commands by the user, the shell supports a command language which it interprets so that complex sequences of commands can be expressed as shell programs and saved in files. TC2000 nX supports two shells – the Bourne shell and the C shell. The Bourne shell, or *sh*, is so named because it was developed by Steve Bourne. The Bourne shell is the common "standard" UNIX operating systems shell. The C shell, *csh* was developed at Berkeley by Bill Joy and has improved user interaction capabilities, including a command history mechanism with recall capability.

switching base

In a Butterfly switch, the number of ports per crossbar switch module. Determines the number of header bits in the routing address of a packet processed by each module.

switching column

In a Butterfly switch, a column of switch modules. It determines the number of input and output ports of the switch. For example, in the TC2000 switch, each switch module has eight inputs and eight outputs. Thus, a two-column switch has two columns of eight switch modules each, resulting in a switch with 64 input ports and 64 output ports.

time-critical

We define time-critical applications to be those applications that demand the most in responsiveness and predictability of the computer system. This includes traditional real-time, hardware-in-the-loop applications as well as transaction processing and decision-support applications where a timely, predictable response is essential. Time-critical applications include applications where the timing must be exact, such as hardware-in-the-loop simulation systems, as well as systems where statistical performance goals (such as average response time) are acceptable.

TotalView

TotalView is the TC2000 system's window-based, source-level, multiprocessor debugger. TotalView is unique in that it allows the programmer to see a complete view of a multiprocessor application from a single debugger session with a fully coordinated window-based user interface. TotalView supports applications running under nX and pSOS^{+m} — even applications that span multiple TC2000 clusters and operate simultaneously under both nX and pSOS^{+m}. The user interface is a straightforward mouse-based point and click interface. To see the value of a variable, or set a break point, the user simply points to the desired variable name in the source, or source line, and clicks the mouse. TotalView is integrated with the Gist performance analysis tool and can set Gist eventpoints just like one would set breakpoints in user applications.

Uniform System

A parallel-processing runtime environment available under the nX operating system on the TC2000 multiprocessing computer. It was especially designed to work with TC2000 C, but can also be used to supplement the functionality of the parallel language extensions of TC2000 Fortran. Several of its major features are that it supports clustering, a shared-memory programming model, dynamic load-balancing for efficient processor utilization, simple mechanisms for data interleaving, and configuration-independent programming.

virtual memory

A system where the user may allocate more memory space for a process than actually exists in main memory. The operating system moves data between a secondary storage device (usually a fast disk) and main memory, as it is required. Such systems provide the illusion of a much larger physical memory. (See physical memory.)

VLSI

Very-Large-Scale Integration. A space- and energy-efficient approach to the design and manufacture of high-performance integrated circuits.

wired-down

Wired-down memory refers to memory that is not subject to paging by the operating system. Thus, once wired-down, the memory will be accessible to the application program without the delay associated with paging the memory in from disk.

The X Window System

A standard, network-transparent windowing system for bit-mapped displays developed by the Massachusetts Institute of Technology and supported by the TC2000 computer.

BBN Advanced Computers Inc.

A Subsidiary of Bolt Beranek and Newman Inc.

10 Fawcett Street, Cambridge, MA 02138
617-873-6000