



Swift Interconnects Ripple the Multiprocessing Waters

For certain kinds of architectures, emerging switch fabric interconnect schemes can enhance multiprocessor systems.

Jeff Child, Contributing Editor

Gone are the days when system designers lose sleep about multiprocessing architecture. Among the market segments that use multiprocessing, there's no longer much debate over which approach to use. Any military design requiring multiprocessor compute muscle embedded in a constrained space—such as an aircraft, radar/sonar system or battleship—uses distributed multiprocessing. Symmetric multiprocessing (SMP) and non-uniform memory access (NUMA) multiprocessing play in the high-end enterprise and scientific computing realm.

With multiprocessing architecture choices mostly fixed within their market niches, the more interesting question is the impact of emerging switch fabric interconnect technologies on multiprocessing. It's unlikely that these interconnects will play a direct role in the processor-to-memory interactions that define multiprocessing architectures. A couple of those interconnects have some technologies along such lines, but it's uncertain whether they'll play an important role.

Enhancing Distributed Processing

The three major approaches to multiprocessing—SMP, NUMA and distributed multiprocessing—differ mainly in the way processors and memory interact (Table 1). Switch fabric interconnects are expected to enhance the overall system func-

Multiprocessing Definitions

Symmetric Multiprocessing (SMP)

In symmetric multiprocessing all the latencies from processor to memory are consistent. CPUs in the system talk to a single memory space. The CPU's caches are snooped, which means there's coherency between the processors. In SMP one processor doesn't get short shrift relative to another in terms of its access to a particular memory space. All memory access are posted to the same shared memory bus. This works fine for a relatively small number of CPUs, but latency problems appear when you have dozens, even hundreds, of CPUs competing for access to the shared memory bus.

Non-Uniform Memory Access (NUMA) Multiprocessing

A multiprocessor system where the latencies are not the same is called non-uniform memory access (NUMA). Non-uniform memory access means that it will take longer to access some regions of memory than others. This is due to the fact that some regions of memory are on physically different buses from other regions. The system is then made coherent by providing hardware to move data close to a given processors as needed. The NUMA architecture was designed to surpass the scalability limits of the SMP architecture. NUMA alleviates these bottlenecks by limiting the number of CPUs on any one memory bus, and connecting the various nodes by means of a high-speed interconnect.

Distributed Memory Multiprocessing

In the distributed memory multiprocessing architecture, processors and memories could be close to each other, but there's no hardware to keep the system coherent. Coherency is maintained by choreographing data movement using software. This requires more rigorous system and software development than the NUMA or SMP approaches.

tionality of distributed multiprocessing. Such enhancements will strengthen the value of distributed multiprocessing, and, in doing so will perhaps eliminate any consideration of moving to cache-coherent multiprocessing schemes like SMP and NUMA.

Some exceptions include where some SMP and NUMA systems play a role in military applications large enough to have a computer room, such as aboard large ships or in military base installations. Also a practical matter, the number of processors in

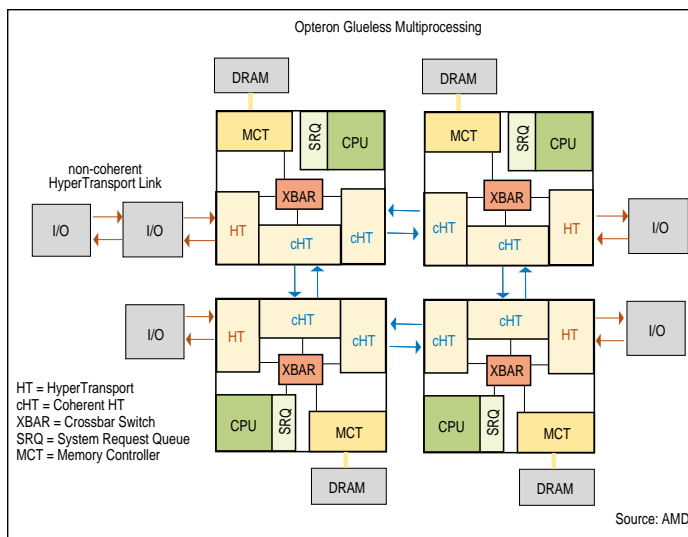


Figure 1 In this four-processor example of an AMD Opteron multiprocessing system, the processors interface between each other and with I/O subsystems over HyperTransport. The HT links between processors use a coherent protocol while links to I/O use non-coherent HT. The crossbar routes command and data info between the memory controller, the HT links and the processors' own System Request Queue.

the system also drives the choice of architecture—SMP gets less feasible as you get beyond around 16 CPUs. NUMA architectures can extend somewhat further, but only distributed architectures are truly suited for applications with hundreds of microprocessors.

Cache Coherency: Who Needs it?

Among the switch fabric alternatives, RapidIO and HyperTransport could in theory fall into the NUMA category of multiprocessing. To use them as NUMA interconnects they'd need a cache-coherent protocol. RapidIO has such a protocol, called the RapidIO GSM (Globally Shared Memory) Logical Specification. The GSM spec follows a globally shared memory programming model preferred in general-purpose multiprocessing systems that support cache coherency in hardware. By making GSM one of RapidIO's overall protocol and packet formats, system designers could marry distributed I/O and general-purpose multiprocessing in the same protocol.

Any practical and economical implementation of RapidIO GSM would require the circuitry for it to reside on a microprocessor—basically an expensive directory-based cache-coherency model would be needed on chip. "It would be challenging to try to do that outside the processor," says Craig Lund, chief technology officer at Mercury Computer Systems, "You'd have to be able to get some hooks into the caches."

Although Motorola offers two processors enabled with RapidIO—its MPC8540 e500 host processor and its PowerQUICC III communications processor—neither chip has GSM support. To date no such processor has been announced. While there's much talk in the RapidIO camp about using GSM in the future, there's a degree of uncertainty around whether the cache-coherent protocol in RapidIO will be important or not.

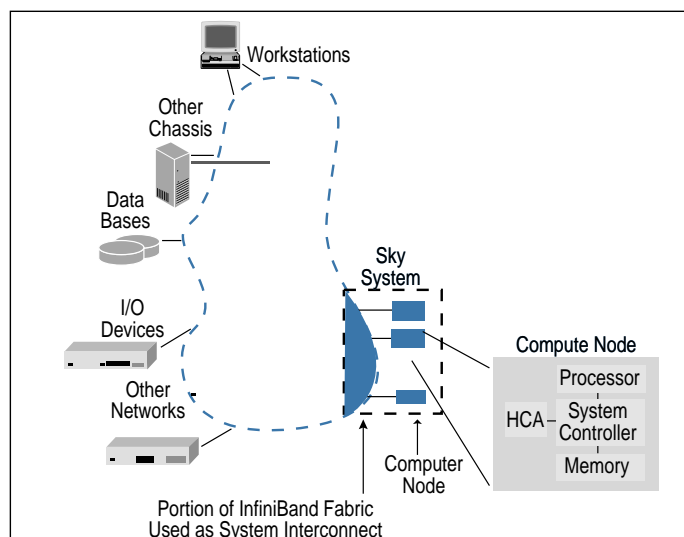


Figure 2 As a fabric in distributed multiprocessing systems, InfiniBand offers a number of benefits such as security. Different nodes in the system communicate using message passing architecture and RDMA. The application determines what memory is available to a remote node.

Meanwhile, there's at least one application of HyperTransport as a cache-coherent protocol. It's not part of the HyperTransport Consortium but rather a spec AMD has crafted on its own for multiprocessing architectures using AMD CPUs. Called Coherent HyperTransport technology, the scheme is used to link multiple processors together where memory coherence is required. An example of processors using HyperTransport coherency is the eighth-generation AMD Opteron processor (Figure 1). Aimed at server systems, there's probably an even higher degree of uncertainty whether this scheme will have any impact in the military or embedded system realm.

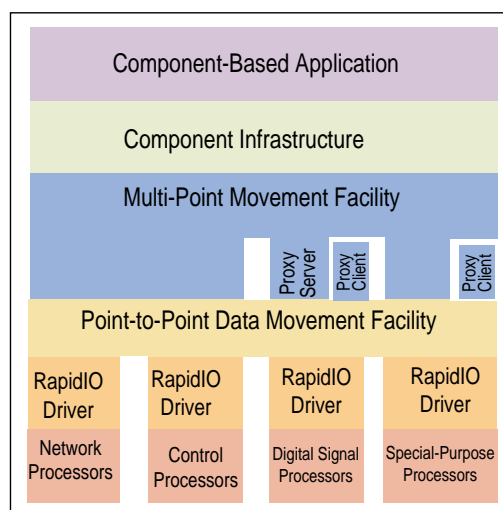


Figure 3 A layered software infrastructure manages data movement in a RapidIO subsystem. The point-to-point facility is responsible for basic data movement between two nodes. The multi-point facility is responsible for complex scatter/gather operations. The component infrastructure provides a standard input/output specification and defines the data partitioning and scaling constraints for components on a RapidIO multi-compute.

COTS View

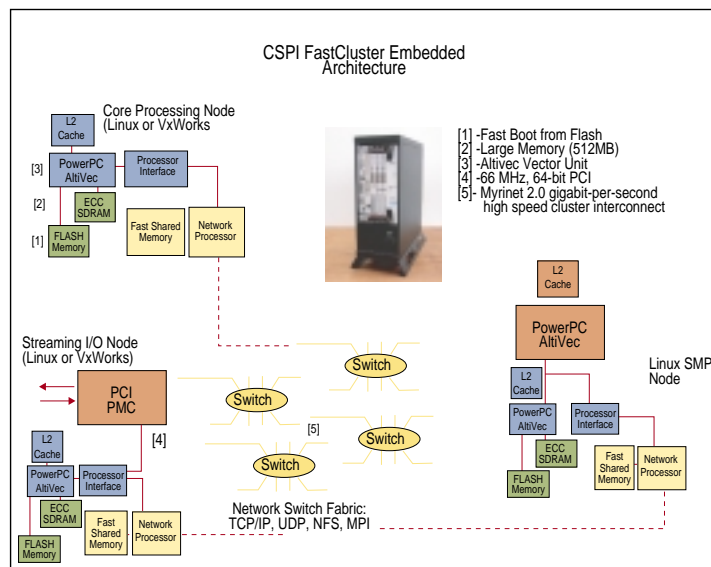


Figure 4 CSPI's clustered processor node approach to multiprocessing leverages the network aspect of the Myrinet interconnects. Several heterogeneous compute nodes can work smoothly together in such a system, including Linux-based SMP nodes, VxWorks-based hard real-time nodes and processor nodes

Continuing down the list of interconnect choices, none of the other alternatives are candidates for cache coherency. PCI Express is clearly aimed at I/O. As a result, it's not focused on cache coherency and will never have it. Technologies targeted for box-to-box use such as InfiniBand, Myrinet, Fibre Channel, GigaNet and StarFabric are all likewise far removed from any cache-coherency duties.

That said, high-speed switch fabrics do enable multiprocessing with the distributed memory approach. The sort of distributed multiprocessing approach that companies like CSPI, Mercury Computer Systems and SKY Computers have taken for years, maps into any of those fabrics. "It's only when you get into the concept of hardware-based coherency that your options narrow and the uncertainty rises," says Mercury's Lund. In a distributed memory multiprocessing scheme each processor can access memory that's nearby or far away, but there's no attempt in hardware to keep those memory accesses coherent. Instead, a distributed multiprocessor architecture leaves it to the programmer to do that.

Fabrics for Distributed MP

Vendors that play in the high-end distributed multiprocessing space such as CSPI, Mercury and SKY Computers are no strangers to switch fabric technology. Each began deploying their respective home-grown switch fabric implementations over 8 years ago. Therefore they have a unique depth of understanding around using high-speed I/O schemes to facilitate distributed multiprocessing and multicomputing. In recent years, these firms have embraced switched interconnect technologies from the commercial world. They each offer quite different views on the available technologies.

For several years, CSPI has integrated the network-based Myrinet technology into its multicomputer systems. Meanwhile Mercury is firmly in the RapidIO camp, and, with Motorola, was a co-creator of the RapidIO technology. For its part, SKY Computers chose InfiniBand as their favorite high-speed Interconnect. Neither Mercury nor SKY has any announced products with those respective interconnect schemes, although such products are firmly in their plans.

Reflecting on why SKY Computers chose InfiniBand (Figure 2) over RapidIO, Steve Paavola, chief technology officer explains that, in contrast to RapidIO, InfiniBand is not shared memory architecture. InfiniBand is a message passing architecture with RDMA (remote DMA) capabilities. With InfiniBand, the memory that's visible to a remote node is only that which the application chooses to make available. That results in a lot of security.

Multiple Program Support

InfiniBand makes it easier for separately developed applications to co-exist successfully, according to Paavola. That's important for the direction in which many military systems are evolving. In the past systems typically ran only one or two executables that were very much tied together. "Moving forward, defense system designers are interested in a more flexible architecture," says Paavola.

"In a radar system, for instance, they don't necessarily know from moment to moment what target they are going to have to deal with and exactly which algorithms they're going to have to execute," continues Paavola, "That means a pool of processors with a pool of capabilities and dynamic reconfigurations on the fly." When applications are reconfigured on the fly, it's very likely for configurations to come about that haven't been tested. The security and high-availability capabilities of a fabric like InfiniBand help ease that.

Although committed to RapidIO, Mercury's Craig Lund sees some value in part of what's emerged from InfiniBand technology. "The whole Virtual Interface concept within InfiniBand is now finding its way into other interconnect protocols," says Lund. InfiniBand supports the concept of Virtual Interfaces (VI) and has a VI Layer. The concept of VI is to put a lot of extra electronics into the network adapter and then trust that network adapter to enforce security, and directly map hardware.

That same concept is becoming popular in the Fibre Channel and Ethernet realms. "It's just emerging and the APIs for VI aren't particularly mature yet," says Lund. "Wrapped up with that is the RDMA concept. Most of these APIs that take advantage of VI hardware also try to expose at least one DMA engine. There's a variety of proposals for that but there's no real consensus about which one will win, if any. I don't know what will happen with it, but it's intriguing."

A model that supports only one RDMA engine would call for some compromises, from Mercury's point of view. Mercury's multiprocessing architectures based on its RACE fabric, and future systems based on RapidIO, make use of a sea of DMA engines. Those DMA engines are used by

the programmer to choreograph data movement. That data movement occurs over a multi-tiered software hierarchy (Figure 3). Data is brought local to processors before it's worked on in most cases. Lund says that issue would have to be addressed before the VI concept could play in Mercury's implementations.

Net-Centric Multiprocessing

For its part, CSPI makes use of Myrinet in its PowerPC-based multicomputers. Myrinet provides full-duplex 2 Gbit/s data rate links, switch ports and interface ports. Its packets may be of any length and thus can encapsulate other types of packets, including IP packets, without an adaptation layer. Each packet is identified by type, so that a Myrinet, like an Ethernet, may carry packets of many types or protocols concurrently.

Myrinet is essentially like InfiniBand in terms of features, in the view of Bernard Pelon, director of product research at CSPI. "But it's an 'InfiniBand' that we started six years ago, so it's more mature and more understood," says Pelon, "For us to transition to InfiniBand would be invisible from a user perspective. They are both networks. We'd have to look at issues of speed, maturity and low power to consider endorsing InfiniBand, but it wouldn't change the architecture of our machines." The network idea is the key, according to Pelon. CSPI's strategy is to enable users to connect a variety of heterogeneous compute nodes (Figure 4) in a system, including SMP nodes and hard real-time nodes, all working together in a system. ■■

AMD
Sunnyvale CA.
(408) 732-2400.
[www.amd.com].

RapidIO Trade Association
San Francisco, CA.
(415) 750-8263.
[www.rapidio.org].

CSPI
Billerica, MA.
(800) 325-3110.
[www.cspi.com].

SKY Computers
Chelmsford, MA.
(978) 250-1920.
[www.skycomputers.com].

HyperTransport Technology Consortium
Sunnyvale, CA.
(800) 538-8450.
[www.hypertransport.org].

InfiniBand Trade Association
Portland OR.
(503) 291-2565.
[www.infinibandta.org].

Mercury Computer Systems
Chelmsford, MA.
(978) 256-1300.
[www.mc.com].



Flexibility: Loosely Coupled MP's Strong Suit

Sharing advantages from both symmetric and distributed multiprocessing schemes, loosely coupled MP offers the best of both worlds.

Mitchell Bunnell, Chief Technology Officer
LinuxWorks

Among the various types of multiprocessing architectures, loosely coupled schemes are sometimes overlooked. Symmetric multiprocessor systems enjoy popularity in the broad computing realm, and distributed architectures are the option of choice in embedded applications. But loosely coupled multiprocessing is worth considering for its advantages in flexibility and scalability.

Certainly symmetric multiprocessor (SMP) systems allow for extremely efficient processor-to-processor communication. Along with transparency to applications, efficient communications is one of the advantages of using that type of system architecture for an MP system. But symmetric multiprocessor systems have some problems. They don't scale well above about four CPUs. Processor scheduling and context switches are both serialized across the processors, causing unavoidable bottlenecks. And the processors must be homogeneous.

Loosely coupled MP systems on the other hand scale much better, avoid the CPU scheduling bottleneck and can be heterogeneous. Treating the different processors in a loosely coupled MP system as network nodes allows for easy and well-understood system administration. But using network connections for application inter-processor communication is not very efficient, especially if some or all the processors share global memory.

By making use of processor-to-processor shared memory and a low level messaging system, it is possible to provide user applications with easy to use system-wide shared memory and semaphores. It is also possible to have a user level messaging system that allows messages to be sent transparent of the transport mechanism—whether it's shared memory, a network, a high-speed serial interface and so on—with no copying of data if the sending and receiving CPUs are the same or they can share memory.

Symmetric Multiprocessing

To understand the advantages of loosely coupled MP systems, it's helpful to explore the two architectures that compete with it: SMP and distributed multiprocessing. Most SMP

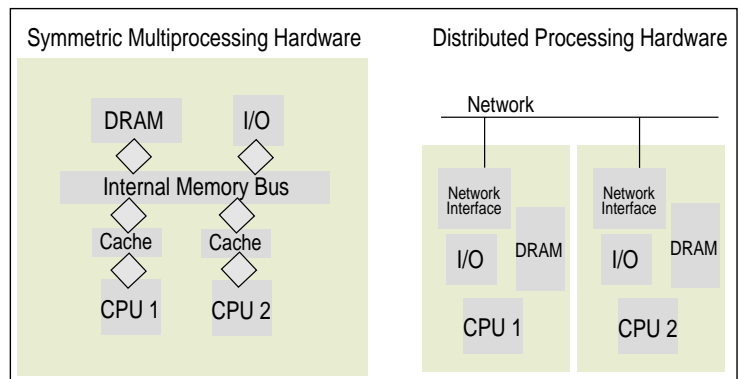


Figure 1 In symmetric multiprocessing (left) all memory is shared by the CPUs. Bus snooping provides memory cache coherency. In distributed multiprocessing (right) each CPU has private memory. There is no memory sharing.

systems (Figure 1) have two or four CPUs. By far the most popular configuration is two. The processors share the same memory and I/O bus. But each processor has its own memory cache. This avoids contention for the fastest and most frequent memory accesses.

The memory caches use snooping to make sure the view of memory is consistent between the processors. Each CPU has a mechanism to identify itself, and each can raise interrupts on the other CPU(s). All CPUs can access the I/O devices. Device interrupts may be routed to one or all the CPUs. If more than one CPU can handle interrupts there is usually a settable mask to choose which processors will receive interrupts from which device.

An SMP system runs a single copy of the operating system, and there is only one copy of the operating system data structures. Tasks (or threads of execution) are scheduled to run on each of the CPUs. Because there is only one shared copy of the operating system, the kernel must use locking mechanism to provide mutually exclusive access to the operating system data structures.

The locking mechanism may be coarse where only one processor can execute kernel code at a time. This was the technique used in early Linux MP kernels. Another technique, which provides better performance, is to use finer locking where parts of the kernel are locked but much of it is reentrant. Also,

different MP locks can be used for different sets of kernel data structures.

Most operating systems allow a single process or executing instance of a program to have multiple threads of execution. In an SMP system, different threads of execution running in the same process can run on different processors. This allows an application program to make direct use of the available processors. To use multiprocessing when parts of a program can be run in parallel, the program simply spawns additional threads to execute the code. When a part of the program is reached that requires serial execution, a mutex protects this region so only one thread executes the code at a time.

Distributed Processing

Distributed processing (Figure 2) is another way to achieve parallelism in program execution. All that is required is a network of computing nodes. Because of the popularity of the Internet, most every computer supports networking. Even inexpensive network interfaces have become very fast, typically 100 Mbits/s or even 1 Gbit/s. A network of computers can have a very large number of CPUs. Routers and switches can be used to make large networks efficient and allow more communication to be done in parallel. In this model there is no shared memory between CPUs, and each computing node has its own private I/O.

In the model of distributed processing via network nodes, each CPU has its own copy of the operating system. Processes do not migrate from one CPU to another. Threads executing in a process all run on the same CPU. Instead of using shared data, programs send messages to one another to communicate. There are many tools for doing network administration. So despite the fact that the CPUs are loosely coupled, booting the whole system, sharing resources and administering the system are fairly simple.

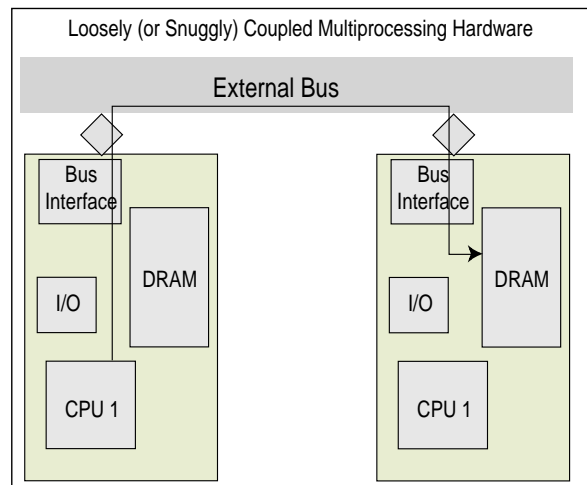


Figure 2 Loosely (or snuggly) coupled multiprocessing systems are a hybrid between shared memory parallel processing hardware and distributed memory parallel processing hardware. The local memory associated with each CPU is shared via the external bus with other CPUs.

Loosely Coupled Multiprocessing

Loosely coupled multiprocessing can be thought of as a hybrid of symmetric multiprocessing and distributed processing. With loosely coupled MP, each CPU has its own private memory. But unlike a distributed processing architecture, this local memory can also be accessed by other CPUs through some type of a system bus. Two popular system buses used for MP systems are the venerable VME bus and the PCI bus.

In order to support loosely coupled MP, in addition to shared memory access over the bus there must also be a mechanism to allow one CPU to raise an interrupt on any other CPU. This can be a simple interrupt mailbox or a more sophisticated message FIFO. For very high performance systems that can support signal processing for instance, the system bus is augmented with some type of a high-speed communications channel. An Example of this is the PowerLine series of processor boards from Thales Computer augments the VME bus with high-speed connections via PCI mezzanine boards.

As with a distributed processing system, each CPU runs its own private copy of the operating system. Processes don't migrate from one CPU to another, and all threads in a process must execute on the same processor. But like an SMP system, processes can share memory and can synchronize without the overhead of sending a message. The model of having an application executing as different threads on the same CPU can be simulated pretty well using global shared memory and global semaphores.

Course and Fine Grain Parallelism

Some types of computing applications can be broken up as rather large chunks of processing work that can be executed in parallel. This is referred to as course parallelism. The overhead of sending a message to another CPU over a network may not be significant to the amount of execution time the other CPU takes to handle the message. In this case it is an ideal job for a distributed processing system.

Examples of ideal applications that exhibit coarse parallelism are a two dimensional FFT and a ray trace graphic rendering program. In the first example each CPU only needs a rather small part of the data to generate a partial FFT, which can be combined with other partial FFTs to produce the final output. In the case of a ray trace, there is a large database of object data. But this database can be replicated to all the nodes at the beginning of the trace. SMP, distributed processing and loosely coupled MP all can handle applications with course parallelism well. But because distributed processing and loosely coupled MP allow the use of more processors than SMP, those types of systems are usually used.

Some types of computing applications exhibit parallelism, but the execution of the application can only be broken up into rather small chunks. This is referred to as fine grain parallelism. Here, the execution requires frequent synchronization of the processors so they can make use of intermediate results. Execution of these types of applications requires that threads of execution be dispatched and joined frequently. An example of an application with fine grain parallelism is a simulator.

Frequent interaction between objects being simulated causes the execution on different processors to need to

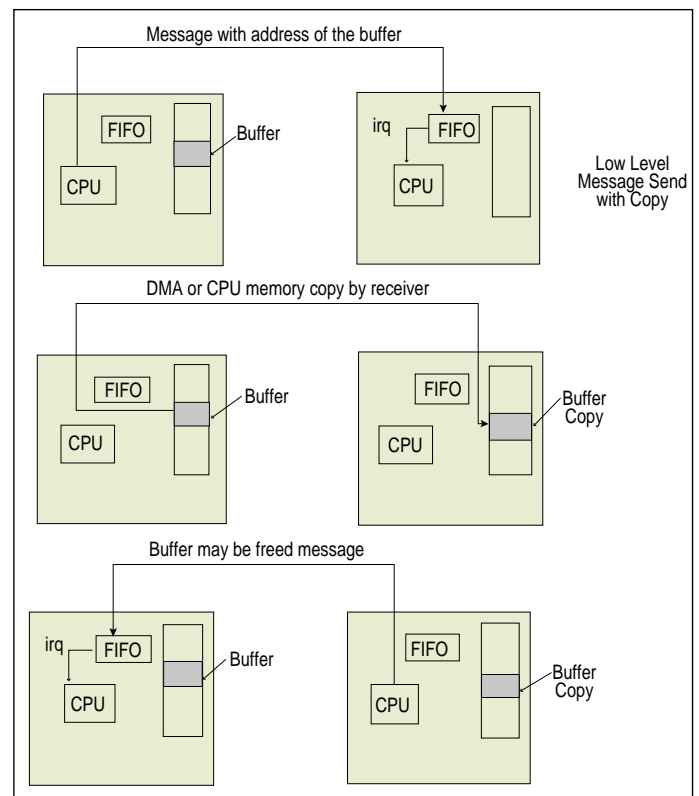


Figure 3 Sending a message by copying is an efficient mechanism for providing a transport for TCP/IP on a loosely coupled MP system. It also works well for multiprocessing application software that has coarse parallelism. The sender notifies the receiver via a write to the FIFO on the receiver. The receiver copies the contents of the message making use of shared memory access to remote memory. After the buffer contents is copied, the sender is notified that the transmit buffer may be reused.

synchronize frequently. Distributed processing systems don't work well for these types of applications because of high messaging overhead. SMP systems are more popular for simulators, but if more than four CPUs are needed to get the desired throughput, a loosely coupled MP system becomes a better choice.

In SMP systems communication is done through sharing a single operating system kernel. In a distributed processing system communications are done by sending and receiving messages through a network interface adapter. In a loosely coupled MP system there is much more flexibility in how the communication between CPUs is done. The technique chosen has a dramatic effect on system performance.

Global Memory and Coherency

Loosely coupled multiprocessor systems can offer applications programs global shared memory that works across CPUs. Requests to attach shared memory sends queries to all other CPUs in the system. A CPU responding to such a query replies with the physical address of the shared memory so the requesting CPU can map the memory in. Global shared memory segments are typically physically contiguous to make the memory mapping easier

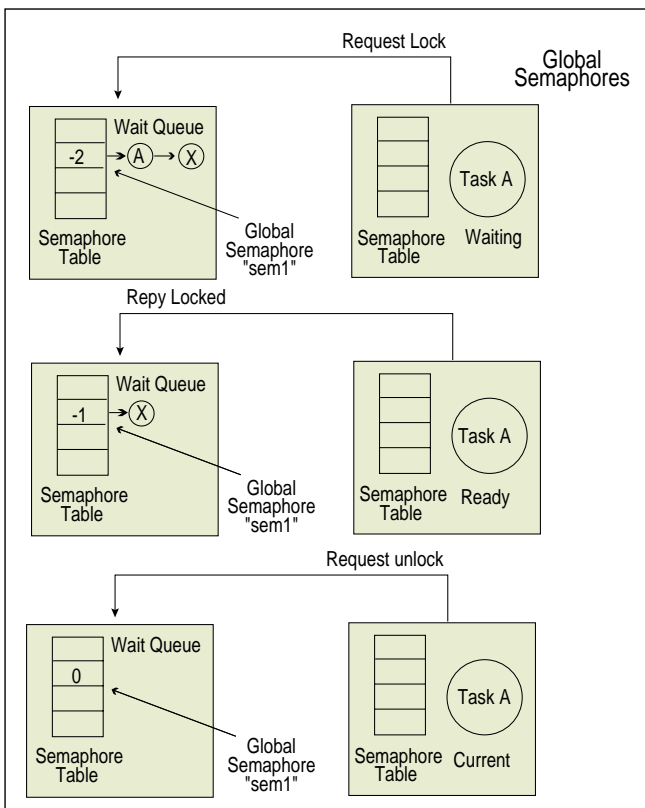


Figure 4 Global semaphores can be implemented using a simple message protocol on a loosely coupled MP system. Each processor has a local set of semaphores but handles messages from remote processors to lock and unlock its semaphores. The combination of global semaphores and user global shared memory works very well for the communication requirements of MP applications with medium grain parallelism.

Cache coherency between the CPUs is an issue for global shared memory. On some systems it is necessary to map the shared memory with no caching enabled. Performance accessing the shared memory is poor in that case. Better hardware supports bus snooping and allows for write through or even write-back caching.

It is possible to provide a global semaphore mechanism that works between CPUs of a loosely coupled multiprocessing system. Global semaphores can be used to provide simple event notification and mutual exclusion between threads and processes running on different CPUs. Global semaphores can have the same standard API as local semaphores such as POSIX Semaphores.

It's fairly simple to simulate an SMP style multi-threaded program on a loosely coupled MP system using global semaphores in conjunction with global shared memory. A segment of global shared memory can be used for all "process data" that is shared between threads. Although POSIX Threads allow for the sharing of data on the thread stacks, very few applications make use of this.

Declaring all global variables as fields of a data structure and allocating the data structure from global shared memory along

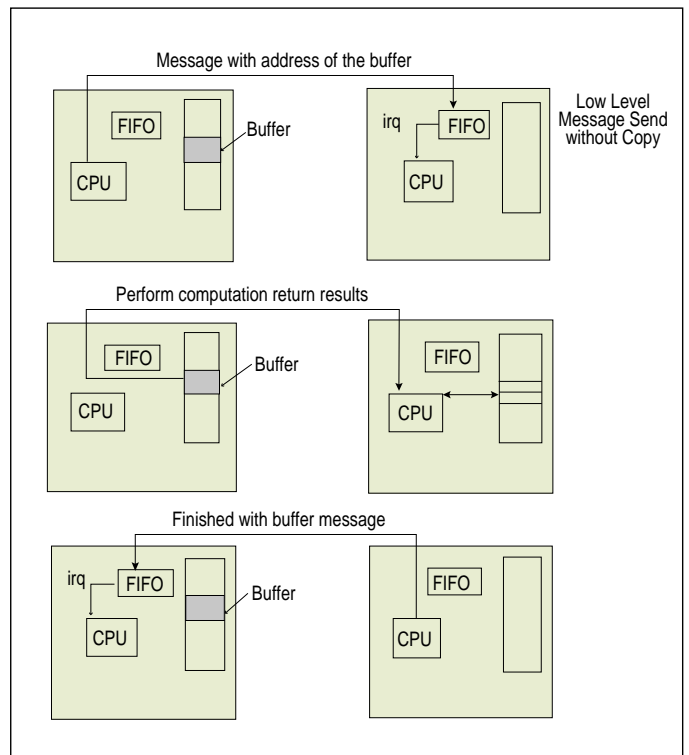


Figure 5 One very efficient mechanism for communication on a loosely coupled MP system is “no-copy” messaging. The “sender” notifies the “receiver” of work to be done on a buffer. The receiver accesses the buffer via the external bus and performs the requested work. When the work is finished the sender is notified. This mechanism works well for some types of applications with even fine grain parallelism. Also, the same API can be used for systems that must copy the data such as those connected through a network interface.

with all dynamically allocated memory is usually sufficient. Global semaphores can be used in place of thread mutexes for mutual exclusion and thread condition variables for event notification.

Transport Independent Protocols

A TCP/IP network layer to support system administration and boot up, plus global shared memory and semaphores is sufficient to support many loosely coupled MP applications. However there is still a reason to support other transport independent protocols. For instance some applications that have already been written to use a different standard messaging protocol such as POSIX Message Queues.

To be able to run those applications in an MP fashion, a version of those message facilities needs to be built that uses the loosely coupled MP low level messaging (Figure 3). It is also worthwhile to provide a facility to support no-copy messaging at the user level. No-copy messaging can be even more efficient than

It is even possible to use loosely coupled MP on hardware designed for SMP.

shared memory and semaphores (Figure 4) for some applications.

An application sends a no-copy message (Figure 5) by first allocating a buffer for the message. This may have to be contiguous memory or have some other special attribute so a special call must be used for the allocation, not just a `malloc()`. The application then sends the message by reference. The buffer cannot be used directly after the send is complete. The application code executing on the receiving CPU uses the data contained in the message to do some computation.

It may write results back to the same buffer. When the computation is complete, the receiver sends an acknowledgment and the buffer can then be reused. The mechanism to provide this interface can be implemented in an extremely efficient manner so that the sending and receiving CPU are interrupted only once and no data copying is performed.

Greater Programming Challenge

Programming an application with fine grain parallelism for a loosely coupled MP system is not quite as easy as programming the same application for an SMP system. However, loosely coupled MP has the advantage that it can support a much larger number of CPUs than SMP. Also, task scheduling, a common bottleneck of SMP systems, is avoided because task scheduling and context switching are performed by each CPU completely independently.

Loosely coupled MP hardware is more expensive than distributed processing hardware because a backplane system bus is more expensive than the physical network layer of a LAN. It is also harder to avoid a single point of failure on a loosely coupled MP system than on a LAN-based distributed processing system. But standard system buses that can support loosely coupled MP are available and inexpensive. It is even possible to use loosely coupled MP on hardware designed for SMP. And shared memory and support for no-copy messaging allows loosely coupled MP systems to support applications with finer grain parallelism than distributed processing systems. ■■

LynuxWorks
San Jose, CA.
(408) 979-3900.
[www.lnxw.com].