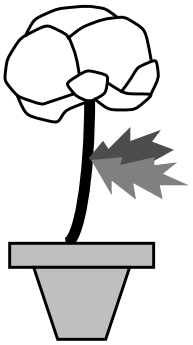
fledgling parallel research projects. Our development continues to make the operating system more robust and efficient using less code (soon under 64K bytes). We are porting to new machines, enhancing the Trollius reputation as a portable multicomputer operating system. Portability means survivability and while the transputer is our bread and butter today, we intend to be around for the next generation of multicomputers.

Trollius is a trademark of The Ohio State University and the Cornell Research Foundation.

UNIX is a trademark of AT&T.

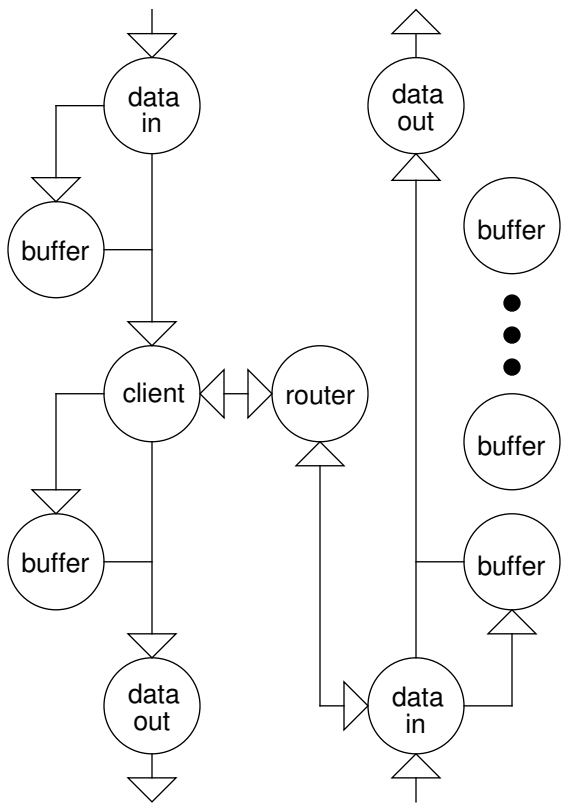


TROLLIUS *portability*

of parallel programming paradigms that can be built on top of Trollius.

On face value, the Trollius network layer would seem to be destined for poor performance due to high overhead. Several compromises, contortions and convolutions in the bowels of the network layer rescue performance numbers but preserve the elegant multiprocess structure.

Together, the kernel and network layer support general multitasking and general process to process message passing in a multicomputer. Our attitude is to focus on this foundation and let others implement their own ideas about parallel programming. Trollius embodies the same characteristics at the operating system level as the transputer, which is an uncomplicated avenue towards parallel processing experimentation.



Multiprocess Structuring - Each node runs a 2K bytes dynamic message rendez-vous kernel supports extensive multiprocess structuring, including the network message passing system shown here. Even buffers are processes.

Remote Services

Trollius uses a simple client/server model to bring every process in the multicomputer the ability to access a UNIX filesystem with the familiar interface routines open(2), close(2), read(2), write(2), lseek(2), etc. When necessary, filenames are prefixed with the node identifier and a colon. Again, we provide the lowest common denominator. Universities and our licensed vendors pick up where we leave off.

For transputer nodes, a server process handles requests to create, doom, spawn and monitor processes. A command that works like UNIX ps lets users know what is happening on the other side of the network. An important goal of Trollius is to dispel the "black box" notion of multicomputer programming and create a more "touch and feel" environment. Trollius has no concept of process placement or compile time maps that distribute an application or data. Since Trollius can also broadcast and multicast messages, we are confident that auto-distribution can be constructed as a higher level service.

Also for transputer nodes, a server process allocates and deallocates memory, providing access through the familiar malloc(3) and free(3) calls.

In general, new Trollius functions are implemented with a new server and a new set of library calls that trades messages with the server. Adding a new server is a trivial chore, another reason why Trollius is suitable for experimentation.

User Interface

As a rule, every remote service accessible through the programming library is packaged into a user executable command program. The user interface is your favourite shell and your favourite windowing system. Trollius does not place the user in a special environment.

Strategy and Long-term Goals

Hopefully, new multicomputer vendors will consider Trollius before investing heavily in mundane, though critical low level system development. We intend to operate in a mode similar to Open Software Foundation, periodically releasing software to our licensees who will then add proprietary value by porting and adding new features yet maintaining basic application portability (we pray). We feel that Trollius will be very helpful to universities with

A View From The Top

Our plan was to be heavily armed with tools before doing battle with a new multicomputer. A goal of the operating system is to equip users to do the same thing. While the transputers are expected to be the major target of application programs, Trollius considers the frontend machine and any machines reachable across a network as equal partners of the multicomputer. A uniform node programming environment, whether the node is a transputer or a Sun, is the underpinning feature of Trollius. In a future version the user will be presented with a menu of every host processor on the network and every installed compute processor (transputer). The user will select a number of nodes, specifying their types (host or compute) and topology and the operating system will boot the same programming environment into every node.

Trollius applications do not require that the user own transputers. An existing Sun farm can be used, for example. When dedicated compute nodes arrive, applications are source code compatible. Since Trollius is multitasking, a parallel system can be simulated in one node.

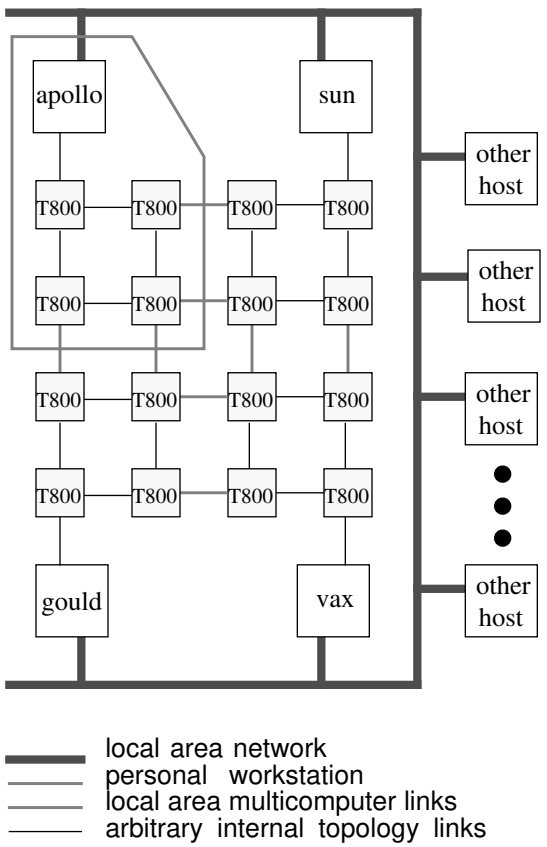
Trollius is not a stand-alone operating system. Every multicomputer running Trollius must contain at least one host node. Trollius combines forces with state of the art multi-user operating systems that have years of experience in standard features like mail delivery, network access and window based interfaces.

Trollius as a Foundation

What is the uniform programming environment on each of the nodes? At the heart of the system is an small kernel whose sole job is to provide a *dynamic* message rendez-vous service. The kernel dynamically creates and destroys processes that communicate amongst themselves on a single node. The kernel leaves scheduling and context switching to the transputer or the host operating system. On the transputer, processes make kernel requests through Occam channels. The host node kernel uses whatever communication mechanism is available to interact with its clients (sockets on UNIX).

The kernel has no knowledge of internode links or the world outside the local node. A group of processes, communicating through the kernel, is responsible for network message passing. Proprietor processes own the internode links and a runtime programmable routing process tells clients which link proprietor process to use. Application programs usually perform all their communication with two calls: nsend and nrecv. However, the network system is like an onion, which can be peeled to lower levels of functionality with less overhead. Trollius also features a virtual circuit capability for use in static communication patterns. In fact, Trollius nicely settles into Occam-style communication when indicated by the application. All message passing calls have non-blocking counterparts and take one parameter, a pointer to a reusable message descriptor prepared by the application program.

Messages are addressed by node identifier and event number, an arbitrary rendezvous value agreed upon by communicating programs. Trollius strives to be very simple; messages are never dropped, acknowledged or timed out. Higher level protocols, protection schemes, and node abstractions are the business



Local Area Multicomputer - Using point-to-point communication, all reachable processors are potential nodes under Trollius.

Trollius: Early American Transputer Software

Gregory D. Burns
The Ohio State University
Columbus, Ohio

While converting Trollius from yet another hypercube operating system called Trillium to a truly portable multicomputer operating system, we have worked hard to make sure that our software is very portable across topologies, compilers, hardware vendors and processors. In spite of processor independence, this article describes Trollius in terms of the INMOS transputer which profoundly shaped the Trollius design. The transputer remains the only processor running Trollius natively today. The success of our project owes much to the success of this microprocessor and good timing with respect to the state of transputer development software on both sides of the Atlantic.

Motivation and Early Design Issues

Trollius was initially developed at the Cornell Theory Center, a US national supercomputer center (development continues at The Ohio State University), because we had an experimental transputer based multicomputer with a Stride (after Sage) frontend running TDS and Occam I. We also had a prospective user community of Fortran breathing scientists and an obvious problem. One of the primary impediments for using multicomputers is rudimentary software. The familiar comforts of home are not often found on this class of parallel architecture. Scientific computational scientists did not want to use Occam I and TDS. While we software engineers had great respect for the language (and even considered a completely Occam based operating system), we simply could not deliver the needed programming environment with the INMOS software of late 1985.

Retooling For Transputer Development

After having ruled out TDS and Occam, we were left with a box full of transputers, no host and no software. Furthermore, we did not have a data sheet for the transputer; the instruction set was a mystery.

With a beta copy of the Compiler Writer's Guide we "inferred" the instruction set well enough to build a transputer instruction set simulator. At the same time, we contracted with a top-notch compiler consultant, Dwight VandenBerghe, founder of Pentasoft Inc. in Seattle, WA. Using the powerful development tools that UNIX provides, Dwight prototyped an assembler and linker from our best guess at the transputer instruction set. The group was now developing and executing approximated transputer code on an approximated transputer simulator. When the official instruction set documentation became available, we emerged into a defined world to find that our guesses were mostly correct. We continued developing other simple flexible tools for the transputer, including a single processor bootstrap loader and a stack based threaded interpreter that served elegantly as a monitor. By the time the C and Fortran compilers arrived, we were equipped with a standard configuration grammar for single transputers that would load any collection of programs into a transputer directly attached to a UNIX machine. The grammar provided for sharing text segments, tuning stack sizes and passing runtime arguments.

tsim	transputer instruction set simulator
tti	threaded interpreter
tas	assembler
tld	link editor
ar(1)	archive object files (also tranlib and tlorder)
tnm	like nm(1)
tsize	like size(1)
tvw	similar to nm(1) and size(1)
tdis	disassembler
tcc	C compiler
tf77	Fortran compiler
moses	bootstrap loader
patch	load module relocator
cboot	single program loader
sboot	system loader

Development Tools - A critical part of Trollius is the underlying toolkit.

All of this work focussed only on single transputer systems. We believe that solid software foundations are laid by starting at the bottom with simple yet powerful tools. The Trollius master plan heavily emphasized layered design with a powerful and complete toolkit as a foundation.