A Local Area Multicomputer

Gregory D. Burns

The Ohio State University
Columbus, Ohio

Expanding Horizons

The Trollius Operating System [1] [2] is expanding the scope of the multicomputer [3] beyond a single box of concurrent processors and a single manager processor. Multicomputer programming methods are applicable and attractive wherever multiple processors can be found, including on the proliferating number of local area network (LAN) connected workstations. Recent development of Trollius has focused on system and application portability on all classes of multicomputer nodes and interconnections.

This paper describes how Trollius portability led to the development of the Local Area Multicomputer (LAM). We present a generalization of the multicomputer concept by expanding the classification of member nodes. The usefulness of a LAM is examined. Implementation is discussed in two parts. First, the existing node portability of Trollius is explained. Then, we discuss at length the recent work towards topology independence, which is crucial to a LAM's flexibility. Finally, we present an early design for machine independent concurrent computation with static load balancing as one method of programming a LAM.

By Any Other Name

Trollius *node*s are not restricted to processors within a box of concurrent computing elements. *Compute* nodes run the operating system natively, but the controlling *host* computers run the *same* operating system and are considered full fledged Trollius nodes in every respect.

Trollius was simultaneously designed for the Unix† multitasking OS and the transputer microprocessor [4] that multitasks in hardware. The development path commenced with the host node and then continued by porting - not redesigning - the existing software to the transputer.

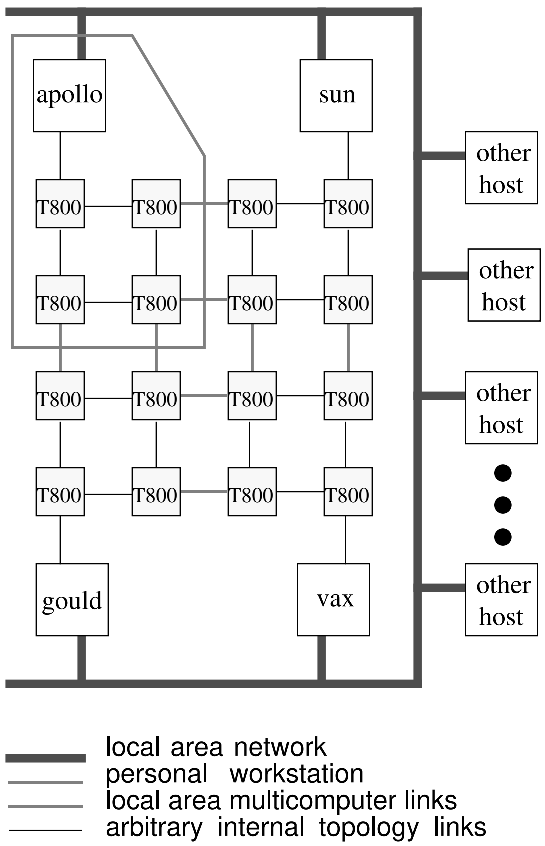
An early idea in the project was to provide for remote access to the multicomputer from the user's personal workstation, also running Unix across a local area network. Since we had already extended the multicomputer to include the host node, we simply extended it one step further to include the user workstation. The same Trollius executables were used on the new node and the transputer link instructions in the compute box were replaced with TCP/IP coupling software. Finally, the Trollius route tables were updated to reflect the extended multicomputer.

The ease of adding nodes on a LAN to a Trollius based multicomputer provided the impetus for the Local Area Multicomputer. With the power offered by unique node portability and the inherent simplicity of the concurrent architecture, a node by any other name (workstation, cube manager, supercomputer, concurrent node) is equally acceptable to Trollius. Any collection of host or compute nodes connected by a LAN can be summoned together as one heterogenous multicomputer running one homogenous programming enviroment. Figure 1 illustrates a flexible LAM configuration.

Uses and Misuses

What benefits does a LAM offer? If it is feasible to operate a distributed multicomputer, is there any point in doing so? Clearly, the load balance of many data decom-

Figure 1



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

† Unix is a trademark of AT&T

position applications is not going to react well to a heterogeneous collection of processors. These problems will realize optimal efficiency and cost/performance on compute nodes only. Yet we foresee five main advantages of a LAM.

- Users may access compute nodes remotely from a personal workstation that would become part of the multicomputer network.
- Compute node applications may take advantage of special devices attached to LAN accessible host nodes.
- Linda [5] class applications exhibit runtime load balancing and may take advantage of compute and host nodes. Trollius provides a Linda-like programming tool called *Brenda*.
- A small multicomputer can be simulated on one multitasking Trollius host node using advanced debugging tools. Any programs debugged for host nodes are source code portable to Trollius compute nodes.
- In the complete absence of compute nodes, many host nodes may be configured as a genuine Trollius multicomputer, again with complete source compatibility to eventual compute nodes.

The Portable Node

Uniform compute and host node programming environments are possible because 90% of the lowest two layers of Trollius source code runs on both node types. In turn, the source code portability owes to the attitude of the kernel and all of the features is does not provide.

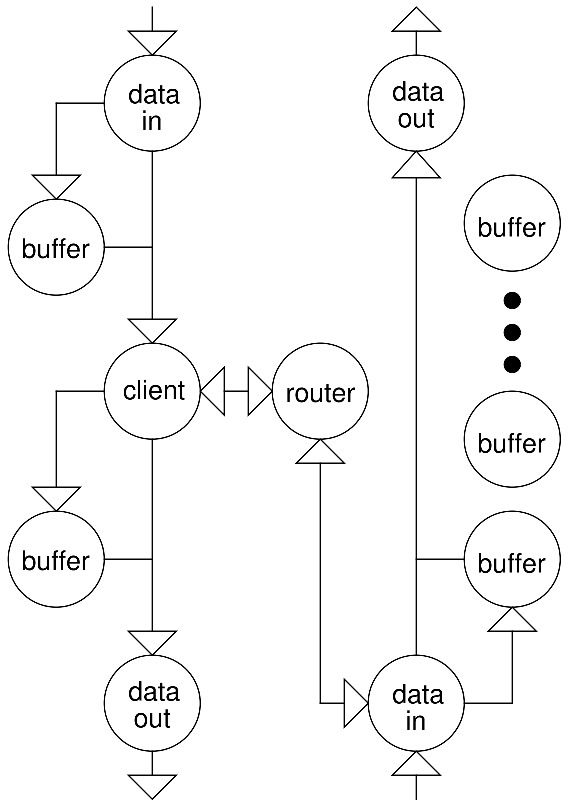
Kernel - The Trollius kernel is uniquely oriented as a rendezvous server, not an executive. The final responsibility for cycle allocation and message transfer resides with software or hardware beneath the kernel. This is intuitive on host nodes since we hardly wish to usurp the host operating system. On compute nodes Trollius runs natively, but the transputer provides the necessary primitive operations in hardware. Thus, there is an ironic similarity in the way both the transputer and Unix, for example, support Trollius.

The kernel distinguishes itself more for the features it does not provide than for the frugal services it does offer. Most importantly, the node kernel does not handle inter-node communication, memory management or complex message passing services such as broadcasting. Its only mission is to pass messages between processes within the node. On a transputer, this can be accomplished with a 1.5 kilobyte code segment. The kernel is the cement foundation on which the portable node operating system is built (and 95% of the kernel source itself is portable).

How would Trollius handle compute nodes not based on the transputer? We believe that message passing research is migrating towards hardware solutions [6]. In place of a hardware solution, we propose a modular software subkernel to fill the gap, just as daughter printed circuit boards are the temporary solutions for VLSI chips.

Network Gearbox - The inter-node message passing system is a multiprocess structured program [7] whose portability is guaranteed by the underlying kernel. Figure 2 is a process schematic of the network gearbox.

Figure 2



Network Portability - By building a multiprocess structured program atop a small message rendezvous kernel, the network gearbox is portable to any conceivable node, either on a LAN or within a concurrent supercomputer

Each bi-directional inter-node link is managed and abstracted by a pair of proprietor processes called datalinks. To send a message to a neighbouring node, a client process sends a kernel level message containing the network frame to the appropriate datalink output process. On input, a client process receives a message from a datalink input process. Only the link-driver code within the datalink proprietors is non-portable.

A router process holds information on every node in the multicomputer. When consulted with a kernel message containing a destination node identifier, the router replies with a message identifying the correct forwarding process. The forwarding process may be either a datalink output proprietor, the destination client waiting to receive a network message, the multicasting daemon, or any other network service we wish to build. The Trollius router is completely portable.

Each buffer is a process, created, monitored and perhaps destroyed by a buffer daemon. Specific buffer management code is sparse, since we simply reuse all of the general process management routines. Buffer quantity, size and creation policy can be tuned with a library or from the command line. Furthermore, buffer monitoring is freely provided by existing process monitoring services. Buffers can communicate (using asynchronous signals) with datalink proprietors to prevent link-clogging without resorting to complicated time-outs or acknowledgments. Buffer software is fully portable.

Though performance is not generally addressed in this paper, it must be mentioned whenever the crucial network message passing software is discussed. We use the analogy of gear ratios on a bicycle or an automobile to describe Trollius communication efficiency. Functionality is paid for by performance and vice versa. Our approach is to provide as many trade-offs or gear ratios as possible for selection by the user. Arbitrary process to process transport, buffering, routing and multicasting are all services which can be peeled off at runtime to lessen overhead. The network gears are not analogue, but we are continuing to add more settings including direct user access to the hardware.

We have explained the portability of the kernel and network layers. The higher level remote and command layers are granted their portability by the network layer. For these reasons, Trollius is highly portable and any combination of processor and memory (even in shared memory multiprocessors such as the Cray Y-MP) can function as a Trollius node with very little effort.

Portable Interconnection Requirements

In the hypercube implementation of Trollius, interconnection software and data structures were either not portable, or programmable only in the laboratory. To support our proposed local area multicomputer, the topology of the nodes allocated to a user had to be programmable at boot time. That is, given a description of physical nodes and physical links, we defined the following requirements for topology portability:

- *routing* - Trollius will route messages on the given topology from a table that is generated at boot time and may be modified at runtime. The previous router was hard-wired for the hypercube.
- *broadcasting* - Trollius will provide a multicast and multireel (inverse multicast) service on the given topology from data structures which are generated at runtime. The application is expected to register the members of the multicast. Multireels are used for global singularities in applications [8] (for example, calculating a global sum) and for broadcast client/server result collection. Previously, Trollius was capable of efficient hypercube broadcasting from the host or root compute node only. Multicasts or reels were not available.
- *booting* - Trollius will boot all nodes on the given topology with arbitrary user assigned node identifiers. Flexible node booting involves the largest and most complex modules in the operating system, yet efficient topology booting was inflexibly tied to the hypercube.

Effective complete interconnection provided by some multicomputers via message passing co-processors (not transputer based machines) would seem to eliminate the need for Trollius management of routing and topology issues. Again, the modularity of Trollius provides for the easy removal of the routing daemon; under this circumstance the entire multicomputer network would funnel through one set of links. However, the scope of a LAM would still require routing and topology support outside of compute nodes.

Programming Nodes and Interconnections

Trollius is very reconfigurable at the node level. Process schema files specify which programs constitute Trollius on each type of node. To add or delete features or whole subsystems, only the process schema need be edited and no recompilation is necessary. Our challenge was to provide a comparable level of functionality at the network level.

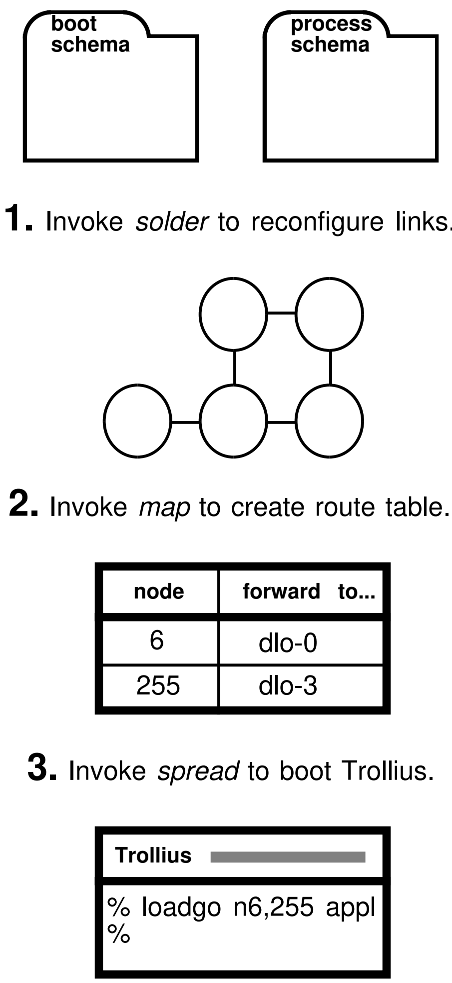
Trollius now provides a grammar called Nodes and Interconnect Language (NaIL) that fully describes the multicomputer at boot time. With NaIL, the user controls the following parameters, which were previously the domain of a Trollius guru:

- *nodes* - Any number of nodes of any type can be identified. Host nodes are referenced by network address. Compute nodes are referenced by network address and a sequence integer (0 - n).
- *node identifiers* - Any unique integer used to label a node (in addition to identification) is presented to constituent processes of the node at runtime.
- *links* - Nearest neighbour connections between nodes are specified. The programmer may supply a weight integer representing the communication load that will affect the outcome of the routing tables.

- *routes* - The path to be used between two nodes may be explicitly given, even overriding nearest direct link connections. Again, a communication load weight may be added.

The information above expressed with the NaIL grammar is called a *boot schema* because it explicitly describes the hardware, node labelling and routing of a bootable multicomputer. The boot schema enables complete flexibility in Trollius's use of installed hardware. Users may own custom boot schema files but many standard topologies are supplied as defaults. Note that Nail is not restricted to describe any single topology class. Trollius tools strive to handle the general case of the arbitrary directed graph of nodes.

Figure 3



NaIL Configuration Tools - The boot schema describes a set of nodes and interconnections allocated to one user. Configuration tools translate the boot schema into effective routing and booting strategies in conjunction with the process schema.

Trollius now provides three tools, depicted in Figure 3, to manipulate the boot schema and create the desired multicomputer:

- *solder* - If inter-node hardware links are reconfigurable, *solder* issues the system dependent commands to connect the nodes as specified in the boot schema.
- *map* - *Map* generates a route table that indicates the forwarding process (essentially the link number) for any pair of nodes. This is a first generation tool based on a few simple heuristics that will certainly undergo much development in the future. In its current incarnation, *map* first creates routes explicitly specified by the boot schema, followed by all direct link connections. Any node pair not already mapped is handled last to cover any eventual multicasting. If a communication load weight is supplied, it is accumulated on each hop of the path. If multiple paths exist, the lightest one is chosen.
- *spread* - Using the generated route table, *spread* boots all *solder*ed nodes in the boot schema with the Trollius operating system.

Next Generation NaIL Tools

The next generation of the Trollius development will build upon the boot schema. Instead of having to supply the boot schema, users will concentrate on describing their own computation. Using the same NaIL grammar, the user will record a *computation schema* and the local system administrator will install a *physical schema*, also using NaIL. The next set of NaIL tools will attack the formidable problem that arises when the computation and the available multicomputer do not match.

The physical schema will describe all installed host and compute nodes that can be booted with Trollius. All possible node pairs which can be connected by a link are also stored. This single physical schema file resides with a Trollius allocation daemon somewhere on the LAN. When processors and links are allocated, a request is made to the central daemon and a slice of the available LAM is carved out, locked and described in a reply to the client.

The computation schema can describe as many *virtual* nodes and links as are needed to solve the problem. Moreover, nodes may be labelled with any number, string or multi-field identifier that best suits the application. As with the boot schema, load weights may be attached to nodes and links. The weights are not only used for route table generation, but also for the mapping of the virtual multicomputer described by the computation schema to the physical multicomputer described by the physical schema. Three tools described in Figure 4, *crunch*, *place* and *detour*, create the boot schema from the previous two schemata.

The computation schema also provides bounds for the number of processors acceptable to run the application. By making a request of the Trollius allocation daemon, the proprietor of the physical schema, the mapping tools discover the target number of nodes, their location on the LAN and their possible interconnection.

Acknowledgments

The Trollius Project is an ongoing multicomputer development effort centered at the Cornell Theory Center's Advanced Computing Facility at Cornell University and the Office of Research Computing at The Ohio State University. OSU graduate students, Raja Daoud and Vibha Dixit are co-designers of the NaIL topology and load balancing tools. Trollius developers at Cornell are David Fielding, Moshe Braner and Jim Beers.

References

[1] Gregory Burns et al, *Trillium Operating System,* Proc. Third Conference on Hypercube Concurrent Computers and Applications, ACM (1988), pp. 374-376

[2] Alison A. Brown, Gregory D. Burns, *The Trillium OS for the FPS T-Series*, Proc. Second Conference on Hypercube Multiprocessors, SIAM (1987), pp. 85-89

[3] William C. Athas, Charles L. Seitz, *Multicomputers: Message-Passing Concurrent Computers*, IEEE Computer, August 1988, pp. 9-23

[4] C. Witby-Strevens, *The Transputer*, Proc. 12th International Symposium on Computer Architecture, ACM (1985), pp. 292-300

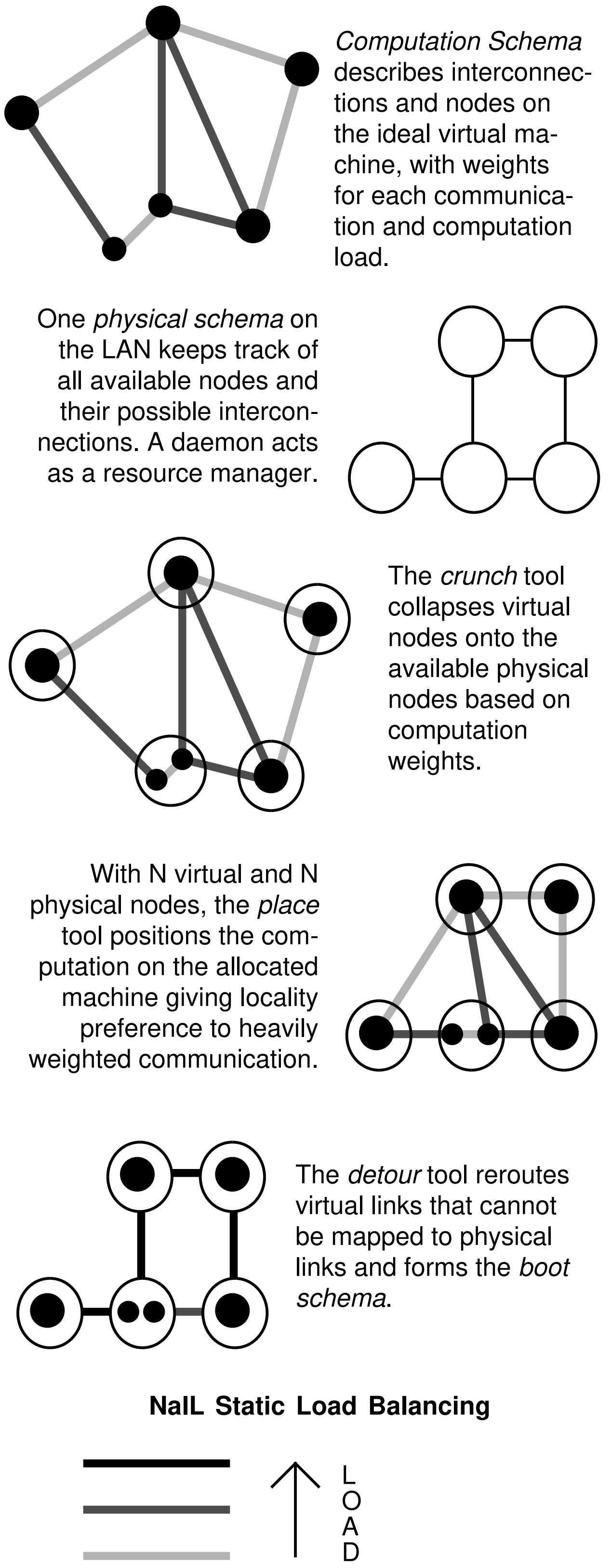
[5] Nicholas Carriero, David Gelernter, *Linda on Hypercube Multicomputers*, Proc. First Conference on Hypercube Multiprocessors, SIAM (1986), pp. 45-56

[6] F. J. Burkowski et al, *A Message-Based Architecture for High Concurrency*, Proc. First Conference on Hypercube Multiprocessors, SIAM (1986), pp. 27-37

[7] D. Cheriton et al, *Thoth, a Portable Real-Time Operating System*, Communications of the ACM, Feb. 1979, pp. 105-115

[8] Karsten Schwan, Win Bo, *'Topologies' Computational Messaging for Multicomputers*, Proc. Third Conference on Hypercube Concurrent Computers and Applications, ACM (1988), pp. 580-593

Figure 4



NaIL Static Load Balancing