Trollius Porting Guide

Trollius is a programming environment for distributed memory MIMD parallel computers, aka multicomputers. This document describes how to port Trollius to an unsupported hardware system. For information on how to install Trollius, consult the *Trollius Installation Guide*. For information on the exact contents of a given release, consult the *Ohio Trollius Release Notes*. Other documents are available which describe how to use and program Trollius.

Ohio Supercomputer Center

Research Computing
The Ohio State University

Document Series 2/9 **March 3, 1992**

How to Use This Document

This document is an informal guide to porting Trollius to a variant of a transputer based, UNIX hosted multicomputer. The reader should not assume that every possible issue has been covered. The first section is a brief discussion of the layout and organization of the source software directory. The second section is the guts of the document, describing the essential steps in developing a porting solution for a new machine. In the final section, an incremental testing plan is suggested.

Names of files, directories and functions are highlighted in **`courier bold`** every time they appear. Important terms are highlighted in *italics* the first time they appear.

The abbreviation ITB means “In The Box” and refers to nodes which run Trollius natively, which for the purpose of this document are transputers. OTB means “Out of The Box” and refers to nodes which run Trollius as an application of the native operating system, in this case UNIX.

Do not hesitate to use the consulting service listed on the last page.

Introduction

Trollius was designed with portability as a primary goal. The first generation software was characterized by the portable node environment - a multiprocess structured networking and remote service system built atop a small, simple message passing kernel. Ninety percent of the kernel is portable.

Where standards in underlying software (UNIX), communication links and node partitioning are to be found, Trollius can easily become portable software. In the absence of standards, Trollius must be ported. Without the assumption of UNIX and transputers as a given standard, this document might not be feasible. Trollius is committed to UNIX but not to any one ITB microprocessor. It is recognized, however, that transputers are the basis of a variety of multicomputer hardware at the date of this writing.

Reconnaissance

The Trollius Project develops, maintains and delivers software in three separate source directories:

LSTT	the Logical Systems Transputer Toolset
Trollius	the essential operating system
Buckeye	the supplementary operating system

LSTT and Buckeye are machine independent. Portability issues are confined to the Trollius source directory. A source directory is sometimes called a *dungeon*. It is important to have a basic understanding of how a dungeon is organized before proceeding with a custom hardware port. A dungeon is divided into the following major subdirectories:

`tools`	toolkit and topology layer utilities
`lib/itb`	ITB libraries
`lib/otb`	OTB libraries
`itb`	ITB programs
`otb`	OTB programs
`share`	ITB/OTB portable code and header files

Dungeons

Figure 1 depicts the basic structure of a Trollius dungeon. The main Trollius dungeon is meant to be machine independent (within the limitations assumed by this document).

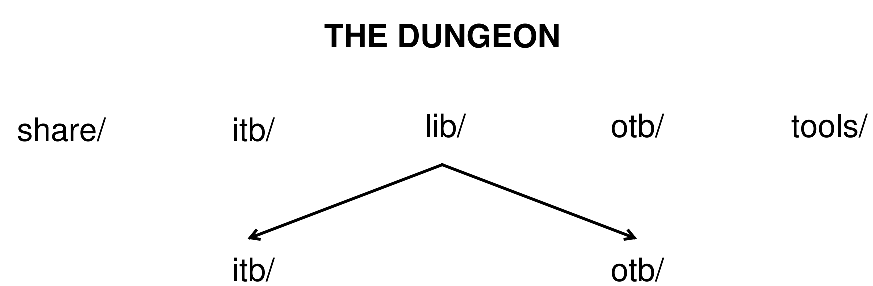

Figure 1

At the root of the main directory is the **`machine`** subdirectory (not shown in Figure 1) and therein are found the machine dependent modules for standard hardware supported by Trollius (see Figure 2). The structure of this software is that of the dungeon in Figure 1. It mirrors the structure of the main dungeon and is called a machine *subdungeon*. Branches or files of the main dungeon that are machine dependent have symbolic links to the matching location in the machine subdungeon. Additionally, the subdungeon contains branches that are not found on all other machines.

MACHINE SUBDUNGEONS

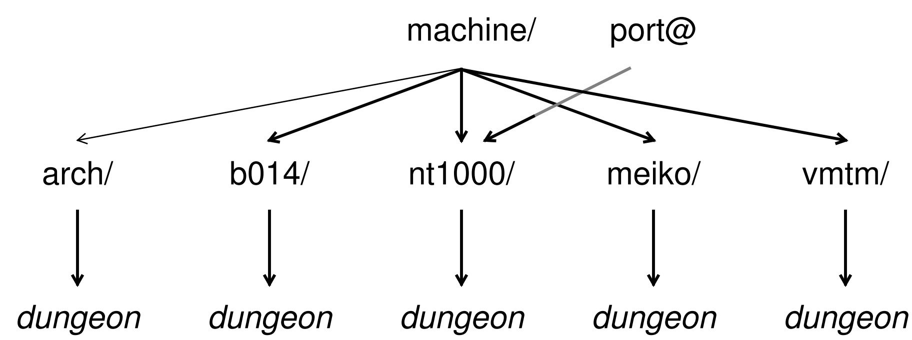

Figure 2

The symbolic link **`port`**, shown in Figure 2 at the root of the main dungeon, points to the active machine subdungeon. The first job in porting Trollius to a new machine is to choose the standard machine subdungeon that best fits the requirements of the new machine. Then choose a new machine name, make a copy of the chosen subdungeon and redirect the port link to the new name. The chosen subdungeon is a useful starting point for implementing a solution for the new machine.

Solution for a New Machine

To port Trollius, one must first find and implement solutions for the following machine dependent features:

- Reset all allocated nodes.
- Program link switches (if present).
- Determine the size of ITB node RAM.
- Establish bidirectional, asynchronous communication between the OTB node and at least one ITB node.

link.c

A large part of the porting effort concerns the interface between OTB and ITB nodes. A quasi-standard module has been defined to isolate specific machine dependent functions. Known simply as *link.c*, it contains several functions which monitor, control and communicate with a group of ITB nodes. A hardware specific link.c (or an object version, link.o) must be supplied for each port of Trollius. The module (source or object) resides in **`port/share/etc`**. The functions contained in a full link.c, and their applicability to Trollius, are listed in Table 1.

FUNCTION	ARGUMENTS	USED IN TROLLIUS
OpenLink	(name)	yes
CloseLink	(linkid)	yes
ReadLink	(linkid, buf, len, timeout)	yes
WriteLink	(linkid, buf, len, timeout)	yes
ResetLink	(linkid)	yes
AnalyseLink	(linkid)	no
TestError	(linkid)	no
TestRead	(linkid)	yes
TestWrite	(linkid)	no

Table 1

Reset

ITB nodes must be reset before they can be booted. There are two strategies available, dictated solely by the hardware. First, a single action initiated from the OTB node will propagate to and reset every ITB node. Second, an action initiated from the OTB node will reset only a device connected ITB node. Remote ITB nodes are reset by their ITB node neighbours. The ITB neighbour reset function is called **`btreset()`** and is located in **`port/`**

itb/sys/bootd/btface.c. Trollius can support either strategy and in either case, reset of a device connected ITB node is the task of the **tfish** tool.

Tfish can be as simple as making the right ioctl(2) call to the device driver. In the absence of a device driver, there is probably a control register mapped onto the OTB node bus that can be programmed. The link.c reset function can also be used. In any case, higher level tools use **tfish** and expect to get a reset of at least the device connected ITB node.

Tfish takes as an argument the name of the device which connects the ITB node. This name is inherited from higher level tools and originates in the TLINK tuple of the boot schema. The argument is necessary because there can be multiple device connected ITB nodes. The name may be a device file name, such as **/dev/link0**, or it can be the numerical address of a memory mapped control register. To Trollius it is a string that is simply passed from the user written boot schema to the machine specific tools.

ITB Link Configuration

Some transputer based multicomputers use programmable link switches to vary the topology of the ITB nodes. These are not intelligent routing switches and are meant to be programmed at boot time and not altered at run time. The TLINK tuples of the boot schema completely describe the link connections between ITB nodes. It is the responsibility of the **solder** tool to set any switches appropriately before boot time and realize the boot schema's topology.

It is also **solder**'s responsibility to establish the reset propagation to all allocated nodes and to isolate the reset from unallocated nodes. This is not relevant to machines with nearest neighbour reset control.

How **solder** does its job is completely undefined. Like reset, it may be accomplished with ioctl(2) calls to the device driver. It may be as elaborate as running a special program on one or more ITB nodes. Methods of link configuration vary widely among vendors of transputer based systems.

The portable portion of **solder** will parse the boot schema and set up all the information in data structures. A set of clearly defined functions must be re-implemented for a new machine. **Solder** is not invoked by any higher level tool. If the target machine does not contain link switches, or if a totally separate software product is already in place to switch links, then **solder** can be forgotten. **Solder** exists to centralize topology issues in the boot schema.

ITB Node Configuration

There are few crucial issues of portability on ITB nodes (where Trollius runs natively) other than RAM configuration. The memory allocation daemon, **`mallocd`**, can support multiple regions of memory. On transputer nodes, there are at least two regions, on and off chip RAM, and this is the default configuration of mallocd, as shown in Figure 3.

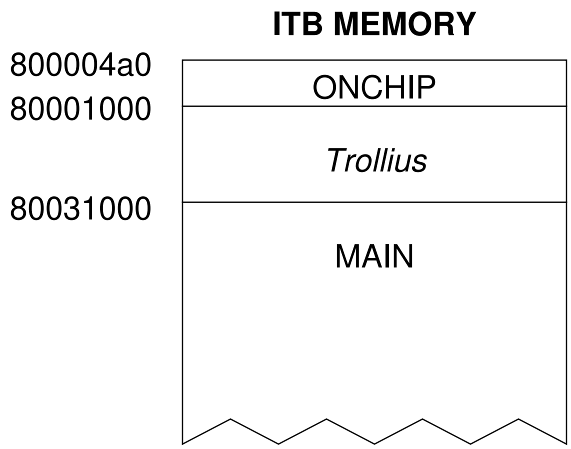

Figure 3

A machine dependent module, **`maconfig.c`**, provides functions that, given a region number, will return the starting address and the size in bytes of the region. For variable size RAM, an auto-sizing algorithm is desirable. This algorithm will depend on how the address space is decoded. In the simple case, reading and writing one megabyte boundaries (a typical incremental size) will eventually turn up an address where the word read does not equal the word written.

Another machine dependent module, **`maconfig.h`**, contains fixed memory addresses for various tools and daemons. These usually do not need to be changed, but the module may be augmented to support the memory sizing and placement functions.

Process Schemata

The programs which constitute the Trollius operating system on any node, ITB or OTB, are described in the process schema. If a change is made to the parameters of an existing program, or if a program is added or deleted, a process schema will have to be altered. Three types of process schemata are described in Table 2.

SCHEMA	NODE	TOOL	PORTABILITY ISSUES
conf.otb	OTB	**hboot**	ITB device names
conf.isb	ITB root	**sboot**	host link program
conf.ivb	ITB remote	**vboot**	none

Table 2

The TOPOLOGY variable in **conf.otb** lists the device names and associated service events representing the links from the OTB node to one or more directly connected, neighbour ITB nodes. The device names are machine dependent and usually must be changed. Unless a special protocol must be observed for communication with the directly connected OTB node, no changes are necessary in **conf.isb**. Finally, **conf.ivb** describes all other ITB nodes and is not changed.

Any process schema may be augmented with support programs for special devices, extra daemons, or anything the local community wishes to become part of the operating system.

ITB - OTB Communication

The most challenging aspect of porting Trollius to a multicomputer configuration of a UNIX host (OTB) node and several transputers is establishing bidirectional asynchronous communication between the OTB node and one or more directly connected ITB nodes. Lack of standards and lack of functionality in UNIX device drivers for transputers is to blame. Several transputer products for UNIX machines have no device driver and polling is the only solution. This section will discuss the best theoretical solution, the best realistic solution and the polling solution.

Trollius Requirements

Trollius has a uniform view of communicating processes running on the two different node types, OTB and ITB. It is not the case that an OTB node will only send a message to an ITB node as a synchronized response to a request for service. Bidirectional asynchronous message passing, as illustrated in Figure 4, is normal and expected in Trollius. The two conversations in Figure 4 are logically concurrent.

There are three types of links in a Trollius Local Area Multicomputer: 1) ITB / ITB, where Trollius is native on both sides and drives the hardware directly, 2) OTB / OTB, using the fully functional TCP/IP and 3) OTB /

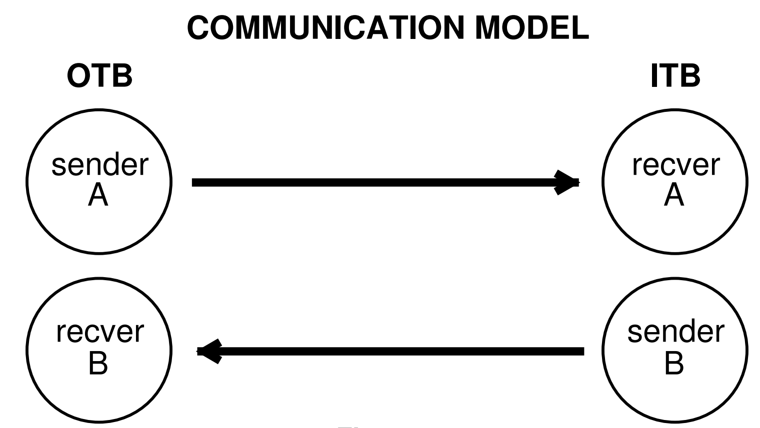

Figure 4

ITB, where a special device driver is needed. The requirements apply to all three, but Trollius has no control and no standard to rely upon for case 3).

Theoretical Solution

The natural and best solution for UNIX is a device driver which becomes part of the kernel. A UNIX kernel interacting driver is necessary to support interrupt driven communication with the device. Interrupts are desirable in a time shared system. In addition, a UNIX driver can provide logical concurrency among multiple process clients. The standard open(2), close(2), read(2), write(2) interface is key to the richness of the UNIX system. A raw byte stream leaves the protocol choice to higher level tools, where it belongs. For such a device driver, ordinary in the UNIX world, Trollius OTB / ITB communication programs require no porting. Unfortunately, most transputer device drivers, when they are to be found, do not meet these standards.

Realistic Solution

If a device driver exists for the target machine, it is likely to fall short of one or more of the points described above. The best generic solution available is in the **`nt1000`** port of Trollius. The **`nt1000`** device driver is restricted to one open descriptor and at most one reader and one writer using that file descriptor. This is the minimum adaptable by Trollius.

The optimal Trollius datalink structure is depicted in Figure 5. An administrator process, **`dl_mom`**, knows the quantity, identifiers and names of all links connected to the local node. For each link, **`dl_mom`** spawns a pair of processes, one input (**`dli`**) and one output (**`dlo`**) to operate bidirectional asynchronous network traffic. To adapt to the sub-optimal **`nt1000`** driver

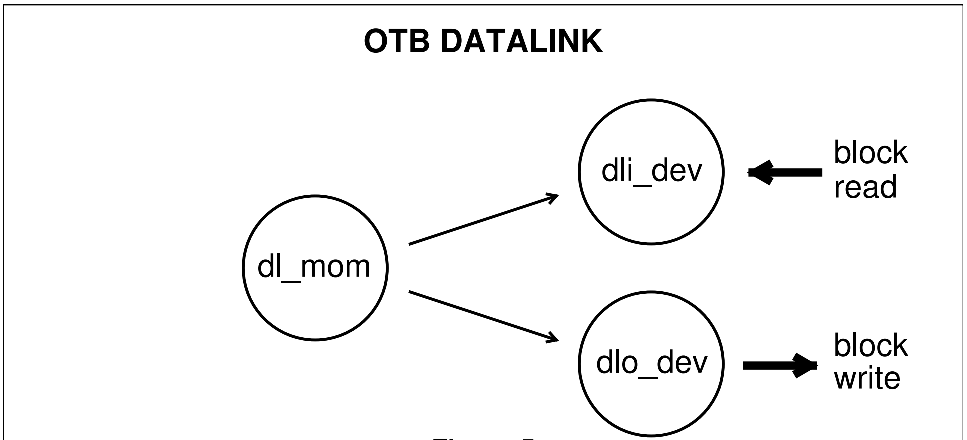

Figure 5

that only allows one process to open a device at a time, **dl_mom** first opens the device and the **dli** and **dlo** parent inherit the open file descriptor.

Apart from the datalinks, which are part of the network layer, OTB device interaction is confined to the toolkit layer. None of these programs require anything more than single process write only access to the device. The **nt1000** driver is completely adequate for the toolkit layer, as most drivers are. All Trollius programs that interact with the OTB device driver are listed in Table 3.

PROGRAM	LAYER	FUNCTION
dl_dev	network	OTB device datalinks
tfish	toolkit	reset device associated ITB nodes
export	toolkit	load bootstrap
moses	toolkit	moses bootstrap front-end
cboot	toolkit	boot single program
thaw	toolkit	load frozen process schema

Table 3

Polling Solution

In the absence of a device driver, or in the presence of a device driver that strictly restricts access to one process, a polling solution must be employed. Although polling is not desirable on a UNIX machine, if the number of log-ins is low, polling is a workable solution.

Figure 6 depicts the OTB datalink structure adapted to a polled device. The key addition is the **dl_idiot** solution which polls both the device for input and a pipe to **dlo_dev** for output. It is not a busy wait. The **dlo_dev** pipe is block read with a timeout which can be adjusted.

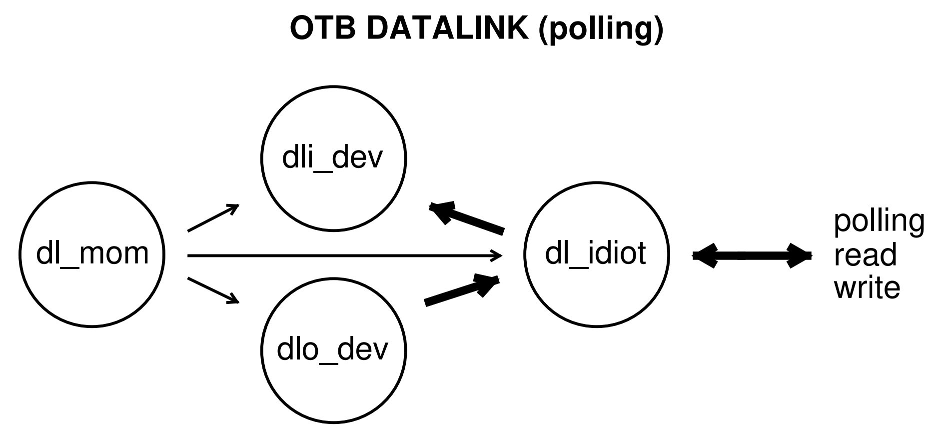

Figure 6

The **b014** machine dependent subdungeon, standard equipment in the Trollius release, uses a polling solution. Access to the device is via the link.c module. The polling solution causes problems for the optimized transputer link proprietor, **dl_t4**, that runs on the ITB transputer node. A substitute program, **dl_ortho**, should be configured into the **conf.isb** process schemata only on the link connected the OTB host node.

Testing the Solution

If the solution for the new machine is implemented and the **spread** command is invoked to boot a multicomputer, there is every chance that Trollius won't work because bugs exist in the machine dependent implementation. A methodical step by step testing plan should be followed. Trollius is best tested in the same manner as it was built, from the bottom up.

- Get moses running.
- Get sboot running.
- Boot one OTB node.
- Boot one device connected ITB node.
- Boot multiple nodes.

A working **spread** does indicate that most Trollius facilities, in particular interprocessor communications, are working properly. Conclude the tests with a check of the file service and Trollius can be declared ported.

Toolkit Layer

The secret to step-wise bottom up testing is devising a test procedure that demonstrates the smallest possible increment in functionality from the currently tested base. It follows that the finer the increments, the easier it is to pinpoint problems. The layered structure of Trollius aids this testing approach.

The most primitive initial test involves **tfish**, the **moses** front-end and the device driver (or equivalent). Reset a device connected ITB node with **tfish** and then run the **moses** front-end. The boot-from-link node should respond to **moses** read and write commands. Next load the ITB **moses.t4** program onto the same node using **export** and test the **moses** read, write and fetch commands.

Continue testing the toolkit layer by loading a trivial program onto the device connected ITB node running **moses**. Carefully choose a base address and invoke **cboot** to accomplish the load and go of the test program. Have the test program write a fixed location in memory which can be inspected with **moses** to see if the program correctly loaded and executed.

Finish the toolkit layer by testing **sboot**, which loads and executes several programs as specified by a process schema. Make a private process schema and remove all but the first Trollius program. Then try **sboot** with progressively more programs until the entire operating system has been booted.

Network Layer

The next step is large because it involves the OTB datalinks, the toughest test of the device driver. Ensure that OTB Trollius is working by booting the stand-alone system (which does not use the device driver) with **`saboot`**. Test basic message passing using **`tping`** or **`ipcdr`**. Check the buffers with **`bfstate`** and the file server with **`fstate`**. Then terminate the stand-alone system with **`sakill`** and boot full OTB Trollius with **`hboot`**. Even without routing tables or node identification (done by **`trout`**), messages can be passed between the OTB node and a single device connected ITB node. Try this with **`tping`** using any node as a destination. Success indicates the hardest part of the porting process has been passed.

Do not attempt all this by running **`spread`**. Find out how things really work by using the lower level tools. After loading the route tables into the two nodes by invoking **`trout`**, further action on multiple nodes requires the assurance from **`solder`** that the links are configured properly.

Boot one additional node with **`vboot`** and load its route table with **`trout`**. If problems are encountered, read the **`vboot`** man page carefully and use **`vboot`**'s extensive options to break down the booting process into smaller steps. Again, test incrementally.

If all is well at this point, **`spread`** can be attempted.

Contact Research Computing
The Ohio State University
1224 Kinnear Road
Columbus, OH 43212

Consulting Phone: (614) 292 8492
Internet: trollius@tbag.osc.edu

TROLLIUS™

Trollius Porting Guide