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 Tutorial Introduction in Fortran

Exit Trollius process:

```
integer errno
subroutine KEXIT(errno)
```

Return local nodeid:

```
integer function IGNDID()
```

Return local pid:

```
integer function GETPID()
```

Send and receive network messages:

```
integer nnode, nevent, ntype, nlength, nflags,
     ndata(*), ndsize, nmsg(*), ierror
subroutine NSEND(nnode, nevent, ntype, nlength,
     nflags, ndata, ndsize, nmsg, ierror)
subroutine NRECV(nevent, ntype, nlength, nflags,
     ndata, ndsize, nmsg, ierror)
```

Appendix A: Quick Reference

Here is a summary of commonly used Trollius commands and functions.

Compile a program for ITB execution:

% **tf77** <source>**.f** [**-ltrillium**]

Compile a program for OTB execution:

% **f77 -I**<idir> <source>**.f -L**<ldir> **-lt** [**-ltrillium**]

Boot Trollius:

% **spread -v**

Check communication:

% **tping n**<nodeid>

Load an execute a program on an ITB node:

% **loadgo** [**-v**] [**-w**] [**-s** <stksize>] **n**<nodeid>
<load_module> [**--** <arguments>]

Execute a program on an OTB node:

% <load_module> [**&**]

Report status of ITB remote processes:

% **state n**<nodeid> [**p**<pid>] [**-g**] [%<index>]

Signal processes:

% **doom n**<nodeid> [**-**<signum>] [**p**<pid>] [%<index>]

Report buffer status:

% **bfstate n**<nodeid>

Clean up message buffers:

% **sweep n**<nodeid>

Become a Trollius process and/or change process priority:

```
integer priority
subroutine KINIT(priority)
```

return without blocking and continue execution. Our send process terminates after sending one message. Where is the message, since it is not with the recv process? It is waiting in a buffer on the destination node (node 1, in our case). Look at the message with the bfstate command.

```
% bfstate n1
```

Bfstate shows a message in a buffer with event 11 and type 0. These correspond to the event and type coded in send.f. Note that the status is "blocked", meaning that the message is ready to be delivered to a synchronizing process. Since the recv process is faulty and will never complete, we will kill it with the doom command.

```
% doom n1
```

In this form, doom will kill all user processes on the specified nodes. Another state command will now show no existing user processes[1]. We also want to remove the message from the buffer on node 1. Use the sweep command.

```
% sweep n1
```

Another bfstate command will show no buffered messages on node 1. It is always advisable to run sweep after a program fails and before running new Trollius programs. Killing a running parallel application will often leave debris messages which can adversely affect subsequent runs.

We now have a clean machine and have demonstrated an important feature of Trollius. It is not mandatory to reboot the multicomputer with each run of a program. You may now repeat the exercise without rebooting.

Terminating the Trollius Session

When you are done with Trollius, terminate the OTB software with the tkill command. Trollius will continue to run on the ITB nodes, but this is inconsequential, since they will be reset prior to another boot. It is a good idea to add the tkill command to your .logout file to ensure that hardware resources are released to other users before you leave.

```
% tkill
```

1. A process ID must be given if state or doom are used with the OTB node.

Executing

To load and execute the send and recv programs on ITB nodes, use the loadgo command. In the example below, we arbitrarily chose nodes 0 and 1 (assuming they exist), but any node can be used that is in the boot schema.

```
% loadgo n0 -v send.itb -- 1
% loadgo n1 -v recv.itb
```

Loadgo will first load the program onto the given ITB node, then allocate memory for the stack and then create the process. The -- option means that all following arguments to loadgo up to the end of the line will be passed as arguments to the created process. Our send program needs the nodeid of node running the recv program. If the processes executed correctly, they will terminate and leave no traces. If you want more feedback, try using print statements within the programs.

An OTB program is invoked from the UNIX shell in the same manner as any other UNIX program. In the example below, we will run the send program on the OTB node, assuming its node identifier is 100.

```
% loadgo n1 -v recv.itb
% send.otb 1
```

Monitoring

The example programs run too quickly to be monitored. Try changing the event in the recv program to 12 (from 11). Recompile the recv program and rerun both programs as before. Now the recv program could not have synchronized with the message from the send program because the events were not equal. Look at the status of user processes on both nodes with the state command.

```
% state n0,1
```

You will notice no processes running on node 0 and the recv program running on node 1, though it is in the state BR (12,0) which means blocked trying to receive a message on event 12 and type 0 - certainly not running. If the process were running, its state would be A for active[1]. The event and type in the status report correspond to the values coded in recv.f.

What happened to the send process on node 0? Should it not still be running, since its message was not received? The explanation is in how NSEND() works; it immediately sends the message along the route to the destination node. Since there are usually buffers along this route, NSEND() usually can

1. BR and A are Trollius kernel states. An active process is a process whose scheduling has been left to the underlying environment. It may still not be running.

Compilation

We will now compile the example send and recv programs for execution on any ITB node. The ITB compiler is called tcc and is used very much like the OTB compiler cc.

```
% tf77 -o send.itb send.f
% tf77 -o recv.itb recv.f
```

The main Trollius library is automatically linked. To compile the programs for an OTB node, the available UNIX f77 is employed, but Trollius header and library directories must be explicitly identified and all Trollius libraries must be explicitly linked.

```
% f77 -I$TROLLIUSHOME/h -o send.otb send.f
     -L$TROLLIUSHOME/lib -lt
% f77 -I$TROLLIUSHOME/h -o recv.otb recv.f
     -L$TROLLIUSHOME/lib -lt
```

After correcting any errors reported by the compiler, proceed to booting the Trollius multicomputer.

Booting

Since the system is composed of independent computers, you must start (boot) Trollius on each node before running applications or using commands. Trollius does not have to be running in order to compile programs.

Boot the multicomputer with the spread command.

```
% spread -v
```

The default boot schema is used. The -v option causes spread to report on the booting process as it progresses. You should return to the UNIX shell's prompt. Trollius presents no special shell or user environment.

Even if all seems well after booting, it is recommended that you check for properly working communication with each node. Tping is a simple confidence building command for this purpose.

```
% tping n0
```

Repeat this command for all nodes or ping all the nodes at once with the ITB broadcast mnemonic, C. Tping responds by sending a message between the OTB node (where the user invoked tping) and the specified ITB node. Successful execution of tping proves that the target node, nodes along the route from the OTB node to the target node, and the communication links between them are working properly. If tping fails, press Control-Z, then reboot the system.

Operation Tutorial

Before using Trollius programs you must establish certain environment variables and search paths for your shell. Add the following commands to your .login or .cshrc (assuming C shell) files.

```
setenv TROLLIUSHOME <Trollius installation directory>
set path = ($path $TROLLIUSHOME)
```

The local system administrator, or the person who installed Trollius, will know the locations of the Trollius and LSTT installation directories. After editing the chosen file, execute it with the shell's source command. This is not necessary on subsequent logins to the UNIX system.

```
% source <file>
```

Many Trollius interface commands require you to specify a nodeid or nodeids. Nodeids are specified on the command line as n<list>, where <list> is a list of comma separated nodeids or nodeid ranges. Example nodeid specifications are:

```
n1
n1,3,5-10
```

Nodeids are established in the Trollius multicomputer plan, called a boot schema, and vary from system to system. During installation, your system will be set up with several common configurations of your multicomputer resource. Learn how the nodes are assigned nodeids in the default boot schema with the following command:

```
% spread -V
```

Some (but not all) Trollius commands allow you to specify all ITB nodes on the command line as C. The fstate command requires you to specify the OTB nodeid. This can be done as shown above using the nodeid learned from the boot schema or with the simple mnemonic, h.

You will also need to identify processes. Processes can be identified in several ways. All process oriented commands will accept p<list> where <list> is a pid or list of pids. Example pid specifications are:

```
p5158
p5158,5160,5200-5210
```

Most ITB node pids can be learned with the state command. OTB node pids can be learned with the UNIX ps(1) command.

The send program specifies the destination node in the nnode argument by either translating an explicit nodeid from the command line or by using the special function, IGORGN(), which returns the nodeid of the origin OTB node. The recv program does not specify the source node when receiving a message. Source node filtering must be accomplished with the message event and/or the type. The nodeid represents the first level synchronization in Trollius message passing. The message first must travel to the correct node before it attempts to synchronize with a receiving process.

The nevent and ntype arguments are used to synchronize a particular message and a particular receiving process within a node. The event is known beforehand to both the send and recv programs. They are often hard-coded but may be learned from the data of previous messages. In our example the arbitrary value 11 was chosen. User events must be positive integers, but there are no other restrictions on the 32 bits. The concept of an event exists in many other multicomputer operating systems under many different names. In Trollius, we try to present it as simply and primitively as possible. It is just a number, period. The event is the second level of synchronization in Trollius message passing. It is not considered until the message has arrived at the correct node. It is considered before the message type.

The message type is a versatile filter that can synchronize in many different ways, as opposed to the strict equality of the event. If given the value 0 by either the sender or the receiver process, it is not considered and only the event determines the synchronization. The type is used as a bit field. If both parties specify non-zero values, then the two values must have at least one bit set in a corresponding position. In other words, if a bit by bit AND of the nh_type fields yields 0 (except in the case of the first rule) the message is not transferred. The type is the third level of synchronization in Trollius message passing. In our example, both send and recv programs set nh_type to 0 to ensure they are not used.

The other arguments used with NSEND() and NRECV() are nlength and nmsg. They describe the length and location of the message to be sent or received.

You should master the workings of nodes, events and types before constructing serious Trollius applications. While powerful, they do not hide the reality of the distributed memory architecture.

Note that both programs, which can be compiled for either OTB or ITB nodes, use standard Fortran program structure, statements, variable declarations and types, and functions.

Initializing, Attaching and Detaching

When a process starts it is unknown to the local Trollius kernel. It must first initialize Trollius related runtime data structures and then attach, or introduce itself to the Trollius kernel. This must be done before any other Trollius functions are called. At the top of each program is the function:

```
call KINIT(0, ierr)
```

which tells the Trollius kernel to include this process in its tables. The argument is an arbitrary integer indicating the scheduling priority of the process. The value 0 is special; it makes the process uncontrolled. The underlying environment will control its scheduling when not blocked on Trollius synchronization. Always use 0 until you understand this topic thoroughly.

ITB processes call KINIT() before entering the main program. A redundant call to KINIT() is harmless and should be done for ITB/OTB portability.

Many functions take an error code as the final parameter. This is a value returned parameter. If the value is 0 when the function returns, the function executed successfully. Otherwise the value specifies the error condition.

The function:

```
call KEXIT(0)
```

causes the process to exit (die) after first detaching from the Trollius kernel. The parameter passed to KEXIT() is the exit code, which by convention should be the error code carried in the global variable errno or 0 if the program completed successfully. Conditional calls to KEXIT() are sprinkled throughout the programs at points where a failure could occur. ITB processes automatically call KEXIT(0) upon returning from the main program.

Message Passing: nsend() and nrecv()

To pass messages between processes and between nodes, the sending and receiving processes each call network message passing functions. Several arguments to these functions describe the message to be transferred and dictate the synchronization point. The calling syntax is:

```
call NSEND(nnode, nevent, ntype, nlength, nflags,
     ndata, ndsize, nmsg, ierror)
call NRECV(nevent, ntype, nlength, nflags, ndata,
     ndsize, nmsg, ierror)
```

```
      if (ierr .gt. 0) then
           call KEXIT(ierr)
      endif
C
C Detach from Trollius kernel.
C
      call KEXIT(0)
      stop
      end
```

The second program receives a message on the same fixed event. Place it into a file named recv.f.

```
C
C Receive a message.
C
      program        recv
      integer*4      buffer(20)
C
C Attach to the Trollius kernel.
C
      call KINIT(0, ierr)
      if (ierr .gt. 0) then
           call KEXIT(ierr)
      endif
C
C Receive the message.
C
      call NRECV(11, 0, 80, 0, 0, 0, buffer, ierr)
      if (ierr .gt. 0) then
           call KEXIT(1)
      endif
C
C Detach from Trollius kernel.
C
      call KEXIT(0)
      stop
      end
```

Programming Tutorial

The examples in this section are very simple Trollius programs that run on any node and pass a message between them. They illustrate the basic objects discussed in the previous section. You will also see how a process begins and ends an association with the Trollius kernel. Prepare the following two programs on your development host.

The first program sends a message to a given node using a fixed event. Place it into a file named send.f.

```
C
C Send a message.
C Read the nodeid from the command line.
C
      program         send

      integer*4       buffer(20)
      integer*4       nodeid
      character*20    argv
C
C Attach to the Trollius kernel.
C
      call KINIT(0, ierr)

      if (ierr .gt. 0) then
           call KEXIT(ierr)
      endif
C
C Fancy paradigms that abstract the nodeid
C are not present at this level. You specify
C the nodeid directly.
C
      if (IARGC() .lt. 1) then
           nodeid = IGORGN()
      else
           call GETARG(1, argv)
           nodeid = IATOI(argv)
      endif
C
C Send the message.
C
      call NSEND(nodeid, 11, 0, 80, 0, 0, 0,
           buffer, ierr)
```

Basic Objects

Trollius does not restrict the interconnection topology of nodes. A globally unique 32 bit identifier is given to each node (nodeid), chosen by the user. Many Trollius functions take the nodeid as their first parameter. The node is the only global object in Trollius. The two other important objects are purely local to each node. They are the process and the event.

Every Trollius node has a multi-tasking, message passing local environment upon which the rest of the operating system is built. There is a small kernel at each node that manages and synchronizes any number of local processes. Each process has a locally unique 32 bit identifier (pid). Many Trollius functions, though not message passing, operate on pids.

The event is an arbitrary 32 bit quantity and the key synchronization point for two local processes. You will see an event in three places.

- in the state of a process blocked on sending a message
- in the state of a process blocked on receiving a message
- in a network (remote node) message

Equal events synchronize. The event has an associate, subordinate object called the type which is found wherever the event is found. It is the secondary synchronization point between two local processes. It is irrelevant if the event does not synchronize. It is irrelevant if it is set to zero. Non-zero types that share a common one bit (bitwise AND is non-zero) synchronize.

munication links exist such that all computers of both types are reachable (directly or indirectly by hopping through intermediate computers). In most cases, general purpose computers, be they workstations or mainframes, are reachable via a Local Area Network (LAN). In Trollius, we refer to general purpose computers as Outside The Box computers or OTB. We recognize the following characteristics of OTB computers:

- They are shared among multiple users.
- Links to other OTB computers are shared among multiple users.
- They may control peripherals needed by a parallel application.
- They will run Trollius as an application of the native, general purpose operating system.

When running Trollius, we refer to ITB or OTB computers as nodes. Very often we make no node type distinction when describing functionality - because there is none. Unless the node type is specified, assume the function applies generally to any node.

Figure 1 represents the broad Trollius multicomputer.

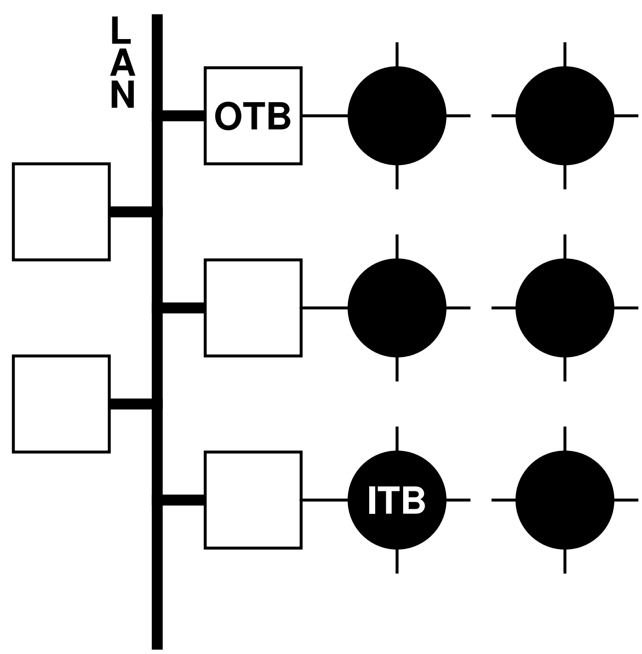

Figure 1: Trollius Multicomputer Architecture

Trollius Architecture

Trollius delivers power to the user through extensive control and monitoring functionality in a highly layered software structure. You can control system details at the lower layers or confine yourself to the comfort and safety of the higher layer. Most importantly, you can decide the balance of performance, functionality and programmer productivity that is best for a particular application. This open architecture design makes Trollius extensible, reconfigurable, and a good foundation for system experimentation.

Foundation is the word which best describes the role of Trollius in the total multicomputer software environment. It is an operating system that focuses on the basics, the nuts and bolts of parallel processing. Compilers, CASE tools for parallelism, and paradigms and abstractions of all sorts need a good foundation. Trollius was not adapted from another computing solution; it was specifically designed for multicomputer applications.

Trollius presents a broad view of the multicomputer architecture. It recognizes that any computer reachable via communication link(s) is a potential participant in a message passing application. The software presents a uniform programming environment across general purpose computers (e.g.: workstations) and dedicated highly integrated computer ensembles. The latter is usually singled out and tagged as the "parallel machine" and is packaged into an impressive looking box. In Trollius, we refer to these computers as Inside The Box computers or ITB. We recognize the following characteristics of ITB computers:

- They are dedicated to single users.
- All incoming and outgoing links are dedicated to single users.
- They are great in number.
- They are going to be used for large parallel computational problems and not the common applications that UNIX workstations do well.
- On average, they have few peripherals.
- They will run Trollius natively.

Since Trollius is not going to provide all the services of a general purpose operating system, the ITB computers must be hosted by a computer with such an operating system. The host computer will have one or more communication links to one or more ITB computers. As indicated above, other general purpose computers may participate in a multicomputer if the com-

How to Use This Document

This document should be your first contact with Trollius. The first section is a narrative introduction to the architecture of the Trollius operating system. The second section is a programming tutorial, presenting and explaining a simple message passing program. Operation of a Trollius multicomputer is discussed in the last section, again in a tutorial format.

This document is user oriented. It does not give much insight into how the system is implemented. It does not detail every option and capability of every command and function. The manual pages cover all the commands and functions in great detail and are meant to supplement this document.

We have kept the font and syntax conventions to a minimum.

`code`	This font is used for things you type on the keyboard or see printed on the screen. We use it in code sections and tables but not in the main text.
<symbol>	This is a symbol used to abstract something you would type. We use this convention in commands.
Section	Italics are used to cross reference another section in the document or another document.

TROLLIUS™

Trollius Tutorial Introduction in Fortran

Trollius is a programming environment for message passing MIMD parallel computers (multicomputers). This document is a tutorial introduction to the operation and programming of Trollius. Only the most basic commands and subroutines are described. *Trollius Command Reference* and *Trollius Library Reference in Fortran* are more complete sources of user information.This document is for Fortran programmers. C programmers are referred to the *Trollius Tutorial Introduction in C*.

OHIO SUPERCOMPUTER CENTER

Ohio Supercomputer Center

THE OHIO STATE UNIVERSITY

Research Computing
The Ohio State University

Document Series 2/5 **March 29, 1992**