TROLLIUS™

The Trollius CrOS III Compatible Library

This document describes the Trollius CrOS III Compatible Library. CrOS III is suitable for data decomposition programs with large data arrays distributed among the processors. Each processor typically runs the same program, and the whole group of processors synchronize at various points in the algorithm, where they communicate. Communication is through message passing in more or less regular patterns.

The Trollius CrOS III Compatible Library, unlike the original CrOS, is implemented for rectangular grids of up to four dimensions.

Advanced Computing Research Institute
Cornell Theory Center

T · H · E
OHIO
STATE
UNIVERSITY

Research Computing
The Ohio State University

Document Series 2/8 **March 26, 1990**

CrOS III

The Crystalline Operating System was developed at Caltech by Geoffrey Fox et al. for the purpose of making multicomputers more easily programmable. The machines they had in mind were the hypercubes built at Caltech/JPL. The applications they had in mind were the data-decomposition methods suitable for physicists. In these programs the data arrays are large, and they are distributed among the processors for parallel processing. Each processor typically runs the same program, and the whole group of processors synchronize at various points in the algorithm, where they communicate. But other than that the processors do not run in lockstep, and may make independent branches in their code. Communication is through message passing in more or less regular patterns, hence the name "Crystalline".

The basic CrOS functions assume that a control program is running on the host, and that this control program loads the parallel program into the parallel nodes, and does the host file I/O on behalf of the node programs. Another programming method is to use a generic, almost transparent, host program, which at Caltech was called CUBIX, in conjunction with special libraries that make the node code that does I/O look almost like standard host (UNIX) I/O code, with parallel extensions. This Trollius library is compatible with the third version of Crystalline, as documented in the book "Solving Problems on Concurrent Processors" by Fox et al. This library does not include CUBIX functions, but Trollius itself provides the CUBIX functionality, albeit with different syntax.

CrOS III Functions Supported under Trollius

All of the functions documented at the end of the book "Solving Problems on Concurrent Processors" by Fox et al. are implemented, except for: *rdstat*, since Trollius does not provide a way to test for a message without picking it up, and the functions *cubeldle* and *cubeldve*, since Trollius does not support environment variables at this point.

This document is primarily oriented towards C programmers, but FORTRAN versions of the functions do exist. See *FORTRAN Programming*.

Implementation of the CrOS III Library

Most recent multicomputers are not configured as hypercubes, but can be configured as rectangular grids of various dimensions. At the same time, most data-decomposition applications only need a grid topology and would not full make use of a complete hypercube topology. Most commonly needed are one-dimensional pipelines, arrays and rings, and two-dimensional arrays, which are called a torus if they wrap-around at the edges of both dimensions.

The CrOS III compatible Trollius library assumes a grid topology, of any dimension (one, two, etc.) up to a limit (currently four). There are certain detailed requirements from the Trollius configuration concerning interconnections and node labeling (see below).

The CrOS message-passing is here implemented via calls to the datalink and network layers of Trollius. These layers allow for long-range communication, which is invaluable for allowing debugging messages to be printed on the user's screen. After debugging is complete, the grid connections may be "welded" into Trollius virtual circuits, removing most of the overhead of the datalink layer, and achieving performance close to that of the physical layer. On the other hand, the usage of the network layer allows emulating a grid that has a higher dimensionality than the possible physical connections, albeit at a lower level of performance due to the message-forwarding involved.

In implementing CrOS communications, Trollius messages use *nh_events* that identify the direction of travel along the grid, i.e., encode the axis index and whether the movement is up or down that axis. The *nh_type* field is used to encode the function being serviced, e.g., *CBCAST* or *CCOMBINE*. It may seem backwards, but this arrangement allows setting up all grid links as simultaneous virtual circuits, provided that there are no clashes between the events modulo 16. The events used by DLO processes must be taken into account, too, and thus the choice of events is non-portable: it currently takes into account the fact that DLO events, modulo 16, are 0,4,8,12. Future enhancements of Trollius may allow event clashes of this sort within virtual circuits, but avoidance of clashes will still lead to the best performance. Note that the event table for virtual circuits is local to the user process.

In the case of the *cshift* function, care is taken in choosing the order of sending and receiving nodes. Although the buffering capabilities of Trollius will handle the case of nodes from both sides of a link sending first and receiving second, the interaction with the buffering subsystem adds overhead. Also, if both receive first, deadlock will result. The algorithm used here tries to maximize the cases where buffering is avoided. To do that we have the even nodes along the shift axis send first, with the odd nodes receiving, then the reverse. This is based on the axis identified by the input mask, which may not have more than one bit set. Care must be taken when the output mask has multiple bits set.

For the functions that communicate amongst all the grid nodes, such as *broadcast*, *combine*, and so on, grid-specific algorithms are employed. (See the source code for more details.) The Trollius facilities for multicasting and multireeling are **not** used here because they cannot function over links that are at the same time taken over by virtual circuits.

Configuring Trollius to use the Library

The physical machine must be booted to support the desired grid, hopefully with direct links. If direct links are missing, the grid will still be supported through message forwarding, but at much reduced performance. And some features will not work at all, such as the "welding" of the grid connections into virtual circuits (see below).

The n nodes must be labeled 0...n-1 so as to fit the grid order, in row-major format. For example, a 2-rows-by-3-columns 2-d mesh has node IDs:

```
        (0)     (1)     (2)     <------- second index
(0)      0       1       2
(1)      3       4       5
^-------------------------------------- first index
```

so that node 3 is at grid position (1,0). Additionally, node 0 should be directly connected to the host.

Runtime errors will result if the node IDs are not 0..n-1, but no errors will result if the physical connections do not fully support the logical grid and message forwarding is thus implicitly invoked.

Program Requirements

The first call must be *gridinit*, in the standard CrOS III syntax. The parameters given to *gridinit* must match the grid that was actually configured (see above). After that any legal CrOS calls are acceptable in any legal order.

Modifications of CrOS III Functions

An unfortunate limitation of this implementation of CrOS is that the receiving and sending of **the same data** cannot be done simultaneously, and neither can a process send data to or receive from more than one channel at the same time. The first limitation is a consequence of the transputer hardware, the transputer being the main target of Trollius at this point. There is no way to ensure that the output data stream does not overrun the input, resulting in garbage being sent. Separate processes can send or receive separate data over separate channels at the same time, but there is no provision for one application process to do that without an explicit fork into multiple heavyweight threads. These limitations do not limit what an algorithm can do, since the desired data transfers can be done sequentially, but the performance will suffer. Therefore, CrOS functions that seem to communicate on several channels at once, such as a call to *cwrite* with more than one bit set in the channel mask parameter, or a call to *cread* with a nonzero output mask, do work correctly in this implementation, but at less than optimal speed.

Channel IDs (mask bit positions) are not the same as they would be on a hypercube. Instead, looking up along grid axis K is channel 2*K+1, down is 2*K. Note that the sender of a message and the receiver of the same message use the same physical channel but do **not** use the same channel masks! In any case, the function *gridchan* should always be consulted for the correct channel mask rather than trying to guess it.

Wrap-around at the edges of the grid is supported by default, but may involve message-forwarding, and thus be inefficient in the case where the wrap-around links do not exist as normal links in the hardware. If an element in the array of grid sizes passed to *gridinit* is negative, it is taken to mean that no wraparound is to be done on that axis, so relevant channel masks will be set to 0. Wrap-around on anything but real physical links does not work under virtual circuits.

The sending functions return once outgoing data is sent out, although outgoing data may not have been received yet at its destination. Trollius currently does not provide for an acknowledgment when messages are forwarded, and the use of separate acknowledgment messages is not deemed worth the overhead. Note, though, that in the case of virtual circuits between all grid neighbors, the sender and receiver do synchronize.

We trust the datalink layer to transfer data error-free, and do not use a checksum of any sort in the code for this library. In the case of transputer links, the data is reasonably secure for short cables, but there is reason to worry if using long cables utilizing electronic or fiber-optical buffering.

No error is returned from any receiving function upon reception of less data than was sent. This is dictated by what can be efficiently done using the existing Trollius calls. There is no *errcnt* variable since we cannot put anything useful in it.

The host processor ID is whatever its Trollius node ID is, not necessarily 1024.

Other minor differences between the implemented CrOS functions and their description in the book by Fox et al. are listed here:

Cwrite returns 0 if the *mask* parameter is 0. It is not clear to me what the original CrOS returns.

Cread returns 0 if the *inmask* parameter is 0. It is not clear to me what the original CrOS returns.

Cread does **not** forward the tail of the message if it did not have a large enough buffer to receive it. This is because Trollius discards that portion.

Cread does not return an error if the incoming data is larger than the specified buffer.

Cread cannot operate unless given an actual buffer. If given *NULLPTR*, it returns an error. This is because the needed memory amount is unknown.

Vwrite returns 0 if the *mask* parameter is 0. It is not clear to me what the original CrOS returns.

Vread returns 0 if the *inmask* parameter is 0. It is not clear to me what the original CrOS returns.

Vread does **not** forward the tail of the message if it did not have a large enough buffer to receive it. This is because Trollius discards that portion.

Vread does not return an error if the incoming data is larger than the specified buffer.

Vread cannot operate unless given an actual buffer. If given *NULLPTR*, it returns an error. This is because the needed memory amount is unknown.

Both *vread* and *vwrite* use *malloc* to allocate a big block to temporarily hold the whole vector in gathered form. If this *malloc* fails, the *vread* or *vwrite* call fails.

Vshift requires two *mallocs*, for input and output gathered forms of the vectors.

Broadcast does not return an error if the incoming data is larger than the specified *nbytes*. The data passed to nodes downstream, though, is limited to *nbytes*!

Broadcast **does** pick up and forward the message directed to the processor even if given a *NULL* buffer pointer, although the data is discarded. Unlike the original CrOS, though, it does not return the number of bytes that the CP sent. Instead it returns the number of bytes received.

Combine allows a *NULL* buffer if *nitems* is 0. Such a data-less combine is useful as a synchronization barrier:

```
combine (NULLPTR, (int (*)()) 0, 0, 0);
```

Concat does not return an error if the incoming data is larger than the specified *binsz*.

Concat requires that the *inbuf* parameter be valid (not *NULL*), and it is assumed that the supplied *inbuf* has space for at least (n * *binsz*) bytes, where n is the number of nodes in the grid. A *NULL inbuf* is an error.

Concat is rather inefficient if *nbytes* is much smaller than *binsz*, since *binsz* is the amount actually passed in messages. The only way to avoid that would be to send each node's data in a separate message. We opted for large combined messages.

Bcastelt does not return the number of bytes that the CP sent. Instead, it returns the number of bytes received.

Combcp does not return an error if *nitems* is too small.

Combelt does not return an error if the incoming data is larger than the specified *nitems*.

In *dumpelt* and *loadelt*, no check is made of the validity of the buffer parameter.

Fcread does not return an error if the incoming data is larger than the specified amount, although it does limit the file output and discards the rest.

The functions *fdumpcp* and *floadcp* use *malloc* to get a buffer for **one** processor's data.

Fdumpcp and *mdumpcp* use *trecv* rather than *nrecv* to avoid message buffer overflow.

Floadcp and *mloadcp* return an error if the number of nodes in *bufmap[0]* does not **match** the actual number of processors, whether it is higher or lower.

Cparam is not void (as is seems to be in the book). Instead it returns 0 (OK) or -1 (ERROR).

Additional Functions

A number of utility functions have been added to those included in the book mentioned above. With the exception of *cweld* and *cunweld*, these additional functions are not intended for use by application programs. Their use is possible but is not normally of any advantage, and they are not portable. Here is a list, including the calling syntax and purpose of each:

```
int   csend (struct nmsg *nhead);
int   crecv (struct nmsg *nhead);
```

Used internally by *cwrite*, *cread*, *bcast*, *combine*, *concat*, etc., to do *psend/precv* or *dsend/drecv* as appropriate. Do not call directly.

`int    cweld (void);`

This requests that all the grid connections be set up as virtual circuits. This reduces communications setup times tremendously. But while the grid is welded no non-local Trollius system calls are allowed. This includes I/O to the host, except from node 0, which can always talk to the host since the link from it to the host is not part of the grid. (The *loadelt* and *dumpelt* functions use direct host I/O from all nodes and therefore are better not used while the grid is welded, although an automatic temporary unwelding is done if they are called.) *Cweld* should not be used unless each and every grid connection, including wraparound if allowed, exists as an independent physical link in the hardware. Call *cweld* from all the grid nodes at once. Do not call on the host.

`int    cunweld (void);`

This call undoes what *cweld* does. It should be used before non-CrOS host I/O, such as *printf*, is attempted.

`unsigned int fullmask (void);`

This function returns the channel mask that is a bitwise OR of all the valid channel masks for the local node internal to the grid.

`int    griddims (int *num);`

This call fills the given array *num[]* with the length of the grid along each of the dimensions. It is used by some of the library functions, and may be used by user-written libraries, to supplement the information returned by *cparam*. But since this is the same information given to *gridinit* as its second argument, this function would not normally be used in application programs.

`int    sendnbr (unsigned int mask, struct nmsg *nheader);`

This call is used by some library functions to translate a destination channel mask to the parameters needed in order to send the message via the Trollius datalink layer calls.

```
int   recvnbr (unsigned int mask, struct nmsg
 *nheader);
```

This call is used by some library functions to translate a source channel mask to the parameters needed in order to receive the message via the Trollius datalink layer calls.

```
int   isoddnode (unsigned int mask);
```

This call is used by *cshift* to determine, given a channel mask, whether the local node is at an odd or even position along the relevant grid axis.

In addition to the functions, there are two global variables used:

```
extern int   errno;
```

This standard global holds the code identifying the most recent error. User programs definitely should look at this -- see below.

```
extern int   cwelded;
```

This global variable knows whether virtual circuits are in effect or not. Holds *HOLD* (nonzero, defined in *<net.h>*) or 0. This variable may be looked at, but not altered, by user programs.

FORTRAN Programming

The FORTRAN functions mentioned in the book by Fox et al are also implemented, but at this point for grid nodes only, i.e., not for the host node. Their names are in uppercase, e.g., *KCREAD*, *KVSHIF*, etc. Note that when calling *KCPARA*, the second argument must point to an array of 6 integers, not 5. Functions added to those in the book include:

KCWELD	same as *cweld*
KCUNWD	same as *cunweld*
KCWDED	returns the value of the *cweld*-ed variable
KFULMK	same as *fullmask*
KGRDDI	same as *griddims*

Porting CrOS III Programs

Programs that assume a hypercube topology would have to be modified. Of particular mention are the extended library functions available from Caltech. We hope to extend this Trollius library with functions ported from the Caltech library at a later date.

Programs that use Parallel I/O (CUBIX) calls would need minor modifications to use the Parallel I/O inherent in Trollius.

Porting Express Programs

The Express system from Parasoft is also a grid-oriented descendent of the original CrOS III, and thus is similar to this library. Many of the calls are the same, but not all. Explicit file I/O from grid nodes is done differently.

Porting Trollius Programs

Many functions in this library make programming typical domain-decomposition applications easier than with straight message-passing. Thus, users of Trollius are encouraged to consider the use of these library functions. One CrOS function will sometimes replace a whole sequence of *nsend/nrecv* calls, and remove the necessity of devising group communication algorithms afresh in each application.

Debugging Programs

Virtual circuits should **not** be used until the application is fully debugged. With virtual circuits off, one can insert a *printf* anywhere in the code, in any node, at any time. Additionally, while the program is running (or crashing), one can query the system on any node from commands on the host, such as *state*, *mstate*, and *bfstate*. In order to interpret the reports from *state* and *bfstate*, one must know what Trollius events and types are used for CrOS messages. The messages use events that identify the direction of travel in the grid. The event encodes both the axis travelled on and the direction along that axis, up or down. The events are set to (4*K+2+S), where K is the index of the grid axis travelled along, and S=1 when the message is going up the axis, -1 when going down. Note that, in a 2-d grid, index 0, the "row" index, is the one that changes **less** rapidly as one goes through the nodes by node ID order. Up the axis means towards nodes with a higher index along that coordinate. In the case where the message is wrapping around the end of the grid, 16 is added to the event, to distinguish it from messages travelling the other way in the case where the wrapping is emulated via message forwarding. Here is a table of events used in one and two dimensional grids:

axis index	going up	going down	overwrap	underwrap
0	3	1	19	17
1	7	5	23	21

Two more events are used for communication to and from the control processor, and are defined in the header file *cros.h*: *CTOHOST* (34) and *CFROMHOST* (38).

The message type encodes the CrOS function involved. Here is the list of types (defined in the header file *cros.h*):

symbol	value	functions
CRDWR	0x001	*cwrite/cread/vwrite/vread/cshift*
CDUMP	0x004	*dumpelt/fdumpcp/mdumpcp*
CLOAD	0x008	*loadelt/floadcp/mloadcp*
CBCAST	0x010	*broadcast* (part of *bcastelt*, too)
CCOMBINE	0x020	*combine* (part of *cweld* and *combelt*, too)
CCONCAT	0x040	*concat*: main data
CCONCATS	0x080	*concat*: size array
CBCASTCP	0x100	*bcastelt/bcastcp*
CCOMBCP	0x200	*combelt/combcp*: normal case
CCOMBCPE	0x400	*combelt/combcp*:error case
CWELD	0x800	*cweld/cunweld*

Finally, in the case when CrOS functions return errors, make sure to inspect the *errno* global variable. It may hold a standard UNIX or Trollius error code (defined in *errno.h* or *terror.h*) or a CrOS-specific error code (defined in *cros.h*), which may be one of the following:

ENUMPROC	601	asked *griddim* for 0 or too many processors
EBADMASK	602	channel mask has too many or few bits set
EBADCHAN	603	channel mask includes invalid set bit(s)
EBADBUF	604	null data buffer

EOVERLAP	605	input and output data buffers overlap
ESIZEDIFF	606	*insize* != *outsize* in self-*vshift*
ENUMPROCS	607	*bufmap[0]* does not fit actual *nproc*
ENOGRID	608	*gridinit* not called
EGRIDDIM	609	grid dimension does not match
EDATASIZE	610	data block too large for buffer or system
EFAKEGRID	611	logical dimension larger than physical

Optimizing Performance

Applications should always try and have the grid "welded" throughout the main part of the program. That means a separation of the code into distinct phases, where file I/O is confined to some phases and grid communications to others.

To reduce message-passing overhead to the minimum, message sizes should not be too small nor too big. The ideal size is *MAXNMSGLEN* (defined in *net.h*, typically 4096 bytes) or somewhat less. Longer messages are broken into packets by the library software. A message that is just over a multiple of *MAXNMSGLEN* requires an extra packet with the full setup time overhead for the transfer of little data. Avoid empty synchronization messages if they can be replaced by the implicit synchronization provided by the data flow.

See Also

"Solving Problems on Concurrent Processors", volume 1, by G. Fox, M. Johnson, G. Lyzenga, S. Otto, J. Salmon and D. Walker. Prentice Hall, 1988, ISBN 0-13-8230226

Contact

Advanced Computing Research Institute
Cornell Theory Center
265 Olin Hall
Cornell University
Ithaca, NY 14853

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

Consulting

At Cornell University: (607) 255-8686
Internet: trollius-consult@tcgould.tn.cornell.edu
trollius-bugs@tcgould.tn.cornell.edu
trollius-info@tcgould.tn.cornell.edu

At The Ohio State University: (614) 292-8492
Internet: trollius@tbag.osc.edu

Copyright