TOPS
A Distributed Operating System Kernel for Transputer Systems

Hubertus Franke and Ben Abbott
Department of Electrical Engineering
Vanderbilt University
Nashville, TN 37235

Abstract

The transputer is a small but complete processor to be used as a building block in high performance message-based computer networks. The transputer hardware supports synchronized message passing and time sliced processes. Programming languages provide transparent access to the concurrency facilities of the chip. However, most transputer programming environments do not support higher level concurrency abstractions. As a result, flexibility is sacrificed since distributed application software is tied to a specific transputer network topology.

This paper describes a programming environment that creates an abstraction level such that a network of transputers is regarded as a virtual machine independent of the network topology.

TOPS provides an extended processes model and a higher level of message passing than those provided by a bare transputer. The development of TOPS arose in the context of various projects requiring a higher level of flexibility.

The Transputer System

The T800 transputer [1][2] is a microprocessor designed specifically for parallel processing. On chip features include: a 32-bit processor, a 64-bit floating point unit, and 4 KBytes of fast RAM. As well, the T800 hardware supports a processes model with time-sliced scheduling. Context switches are performed in hardware (overhead is sub-microsecond). Low level message passing primitives are used for intra-processor communication. These primitives support synchronized message passing through channels. A channel is a specific memory address used to control interprocess communication. A process is suspended automatically if the corresponding communication partner is not ready. Upon sender-receiver synchronization, the message passing is performed by a block copy from the source to the destination process. After this, both processes are allowed to resume their computations.

Any two transputers may be linked together via one of four bidirectional serial communication lines operating at a rate of 20MBits/sec. The channel concept is supported for inter-processor communication. This is done simply by mapping a channel address to one of the serial link addresses. Reading or writing the link address accesses a special memory mapped, instruction overlapped DMA controller that provides the message passing function. Only one channel may be mapped to a specific link address. Thus, without special software inter-processor communication is limited to the four neighbor processors.

Functionality

Although the transputer was designed for OCCAM, OCCAM has not gained general acceptance from the transputer users community due to: (1) portability issues, (2) its lack of support for any dynamic nature and (3) its lack of structure. TOPS has been developed from the concurrency routines [3] of Logical System's Parallel-C (LSC) [4].

The following functions are supported in TOPS:

Task Handling: A task is identified by its task number and its processor identification. Typically, a task is...
implemented as a server process receiving messages through various ports. Any task in the system can send messages to a port. However only the task owning a port may receive messages from it.

**Message Passing**: Tasks communicate with each other using high level message passing routines such as SendMsg, RemoteProcedureCall or Broadcasting. The underlying kernel automatically invokes interprocessor communication if tasks are allocated on different processors.

**Event Handling**: A set of events can be assigned to a process. An event can be signalled from anywhere in the network. The owning task can wait on an event.

**Fast Memory Management**: Fast memory management has been implemented to utilize the dynamic memory requirements of the message passing and to reduce existing overhead.

**Debugging / Resource Supervision**: The kernel can be operated in supervised mode, that is addressed tasks, ports and events are examined for their existence, messages which can not be delivered are reported etc.

**Building Utilities**: Functions to create tasks with their associated ports and events, specification of the routing information, etc.

These functions are provided in the form of the TOPS kernel on each processor. They allow the programmer to consider the transputer network as a virtual machine.

**Task Model in TOPS**

TOPS uses the term task rather than process. A task is similar to a transputer process, but has a few extensions. Tasks communicate only using higher level message routines. A task consists of:

1. code to be executed,
2. workspace,
3. message ports,
4. events to synchronize multiple tasks.

Tasks are created on a processor using the builder routine:

```c
CreateTask (int task_id, int (*code),
            void *workspace, int workspacesize,
            int port_nr, int event_nr,
            int param_nr, ...)
```

**Communication Model in TOPS**

The use of channels as a communication medium for inter-process communication has the following disadvantages:

1. Only synchronized communication is supported.
2. The message size must be known by the sender and receiver.
3. Only processes on the same or on adjacent processors can communicate.
4. Only one channel can be mapped to a hardware link.

This introduces constraints on how tasks can be placed on different processors.

The TOPS's communication model overcomes these problems. In particular asynchronous message passing between any two tasks in the system is allowed regardless processor allocation. The communication medium is based on ports. A port is a message queue which stores incoming messages in their order of arrival. The size of the queue can be limited at creation time. A port is owned by a task. Only the owner may receive messages from a particular port. Queue entries take on the data type `Message`. Message keeps the command token of a message, its parameters, and, if necessary, information about the sending task. Access to the different members of this structure is hidden through various macros.

The following message passing functions are provided in TOPS.

```c
SendMsg (int node_nr, int task_id, int port_nr,
          int cmd, void *data, int size);
```

`node_nr` specifies the processor the destination task with `task_id` is allocated on, `port_nr` specifies the port index of the destination task, `cmd` specifies the command token, `data` the address of the parameter block and `size` the size of the block. `SendMsg` is nonblocking, that is the calling task continues immediately.

```c
int RemProcCall(int node_nr, int task_id,
                int port_nr, int cmd, void *data,
                int size, void result);
```

Issues a remote procedure call. This function is blocking and returns only after the destination task has replied. It requires all of the parameters of `SendMsg` as well as an address of a buffer to store the remote procedure call result. The function returns an error code concerning the success of the remote call.
**Event Handling**

Most of the applications implemented upon TOPS required some form of task synchronization. For this purpose, events are supported in TOPS. A task can own a set of events. An event is a counter which can be initialized globally, decremented by any task and checked or waited on by the owning task. The event is considered to occur when the counter reaches zero. After that, the counter is reset. Synchronization using message passing is in general possible but has the disadvantage that these messages are queued and that the counter has to be explicitly implemented. The following routines are available for event handling: `SetEvent`, `TestEvent`, `SignalEvent` and `WaitEvent`.

**Dynamic Memory Management**

The message passing routines use dynamic memory for allocating storage blocks to store the parameters and build up the message queues. Once a message has been forwarded or has been processed, the occupied resources have to be released. Most run time systems use a heap for dynamic memory management (DMM) providing the functions `malloc` and `free`. It is known, that these functions are not very fast. TOPS provides a table based DMM above these functions. Memory blocks of the same size are stored in a linked list. Allocating memory is popping the appropriate list, freeing dynamic memory is pushing the block on the list. When no memory is available for allocation the appropriate list, freeing dynamic memory is pushing the block on the list. When no memory is available for allocation a garbage collector is invoked. The table is configurable. The length of the table as well as the memory block size can be specified. Their product determines the maximum number of bytes that may be allocated through this fast method. Any larger request is mapped into the general system functions. This table-based DMM has been proven to significantly improve the performance of the message passing. For instance in typical communication intensive applications an average performance increase of 30% has been measured.

**Input-Output in TOPS**

The I/O routines in LS-C [4] can only be invoked on the root transputer. In TOPS these routines have been extended to support secure I/O on each processor. On each processor an IO server is provided which communicates to the host server program. On the root processor, this communication follows a LSC fixed protocol where
from any net processor remote procedure call is used to communicate with the host.

**Evaluation**

Performance analysis of message-based parallel processing systems proves to be a difficult task [5]. When targeted for real-time systems two parameters are of primary interest:

- *Transport rate*, the communication data per second supported by the underlying hardware. In the case of the transputer, this rate is 1.6 Mbytes/second for a single direction transfer from one processor to one of its neighbors. This rate drops to 1.25 Mbytes/second (each way) for a concurrent bidirectional transfer. Thus, the total bidirectional bandwidth between two transputers is 2.5 Mbytes/second.

- *Overhead* is the time involved with the transmission and reception of messages. This time consists of buffer management and hardware transfer initiation.

If the underlying bandwidth of the hardware transport mechanism is to be efficiently utilized the ratio of overhead / time to transfer a data packet needs to be low. If all the messages of an application system are sufficiently large (ie. overhead is much less than the time to transfer one message) the message throughput will be equivalent to the hardware transport rate. The interesting thing is not the fact that larger messages will have better efficiency. Rather, a curve depicting efficiency vs. message size is of interest. Knowing the Overhead and Transport rate allows a reasonable message size granularity to be established.

Figures 1 and 2 display the results from measurements made in order to characterize the message passing facility supported by TOPS. Eight test cases are shown:

- Case I consists of two processes A and B on adjacent processors. First A sends a message to B and then B returns it. Repeat. This is very much like a remote procedure call.

- Case II consists of four processes A, B, C, and D. A and B are on adjacent processors. C and D are on the same processors as A and B respectively. A sends a message to B and then B returns it. At the same time, C sends a message to D and D returns it. Repeat. Thus, this is like concurrent remote procedure calls.

- Case III consists of two processes A and B on adjacent processors. At the same time A sends to B and B sends to A. Then, each receives its message. Repeat. This allows bidirectional overlap.

- Case IV extends Case III with processes C and D in the same way Case II extended Case I.

- Cases V to VIII are the same as I to IV with the exception of all processes always being placed on the same processor. This tests intraprocessor communication.

The results show the reasonable message size for interprocessor communication to be somewhere greater than 200 bytes. Case I can never reach the same throughput as cases II through IV since it is half duplex. The hardware limit for case I is 1.6 Mbytes/second. The hardware limit for cases II through IV is 2.5 Mbytes/second.

The results for intraprocessor communication tests show the performance in all cases under 1000 bytes to be limited mainly by the overhead. This is understandable since the transport rate in this case is the rate of the local memory bus (limited at 8.6 Mbytes/sec).

**Conclusion**

This paper introduced a distributed operating system kernel for transputers, TOPS. TOPS has been successfully used for implementing a distributed robot simulation system, a remote graphics library and diverse smaller projects. In this context, the concepts of TOPS have proven to be very helpful development tools for distributed software.
Acknowledgement

The first author would like to thank IBM T.J.Watson Research Center for the opportunity to perform the described research during the 1989 summer internship. The support of D. Shea and L.C.Zai (IBM) throughout the project is gratefully acknowledged.

References