TRANSPUTER NETWORK CONTROL

To load an application from a host to network, the network must be connected to the host. The transputers in the network must be interconnected via links; the network topology must match that described in the application configuration.

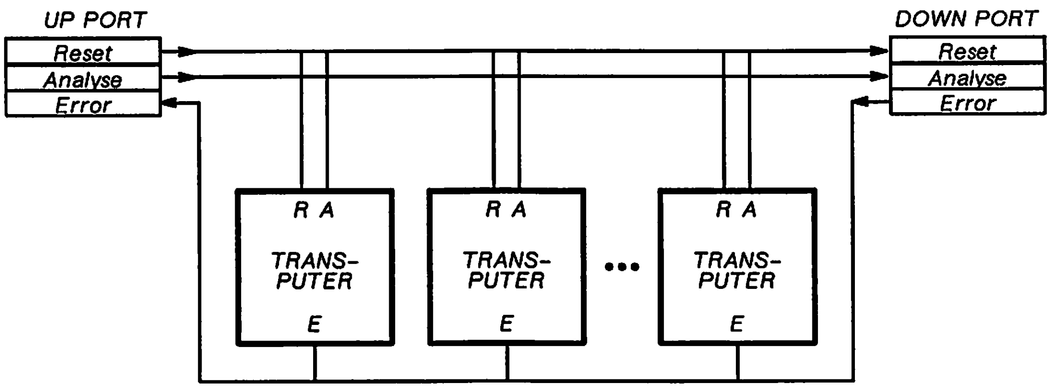
System control signals to monitor and control the state of the transputer network:

Reset: *signal from the host to the network, resetting all transputers, ready for loading.*

Analyse: *signal from the host to the network, bringing all transputers to a controlled halt, so that their states can be examined.*

Error: *signal from the network to the host, indicating that one of its transputers has set the error flag.*

A network consists of daisy-chained boards, containing transputers. Typical board layout (links not shown):



TRANSPUTER NETWORK CONTROL (continued)

To load an application from a host to network, each processor must be loaded with an appropriate code and started up.

The problem: The only route to each processor on the network is via other processors (starting from a root processor, connected to the host). Therefore each processor must first aid in routing the code to other processors, before starting to run its own code.

Bootstrapping a transputer:

After reset, the transputer waits for a communication along any of its links. Having detected incoming data, it interprets the data according to the value of the first byte. Possible cases:

1. If the value of the first byte is 0, *then two words are input. The first word is treated as an address, the second word is considered data to be written ("poked") to that address.*

2. If the value of the first byte is 1, *then a word treated as address is input via that link, and the contents of the memory at that address is sent down the corresponding outgoing link. (Such "peeking" of transputers is useful in the network analysis and exploration).*

3. If the value of the first byte is 2 or greater, *then the incoming data are treated as a code packet, of length specified by the value of the first byte. The transputer copies the code packet into memory (starting at* `MemStart` *location) and begins to execute this code. ("Bootstrapping").*

TRANSPUTER NETWORK CONTROL (continued)

Network loading sequence:

1. *Reset the network;*
2. *Send a bootstrap program packet to the root processor;*
3. *The root transputer starts:*
    a. *The bootstrap program initializes the transputer;*
    b. *The bootstrap program receives the series of code packets, being the code of the* **distributing loader,** *and places them in the memory above its own code;*
4. *The root transputer starts running the distributing loader:*
    a. *The distributing loader sends bootstrap packets to neighbouring transputers, starting them;*
    b. *The distributing loader, directed by the host, sends the code of itself to the transputers it just started;*
5. *Each transputer running the distributing loader starts its neighbours as in (4) until the whole network is started and all processors run the copies of the distributing loader;*
6. *The host sends the application code packets, with prepended processor address information (depth first):*
    a. *The distributing loaders running on the root and intermediate processors use the processor address information to forward the code;*
    b. *The distributing loaders running on target processors copy the application code into memory and make their processors start running the application code;*
7. *The entire procedure of step (6) is repeated (at reduced depth), until the whole network runs the application code.*

TRANSPUTER NETWORK CONTROL (continued)

Network loading algorithm – implementation considerations:

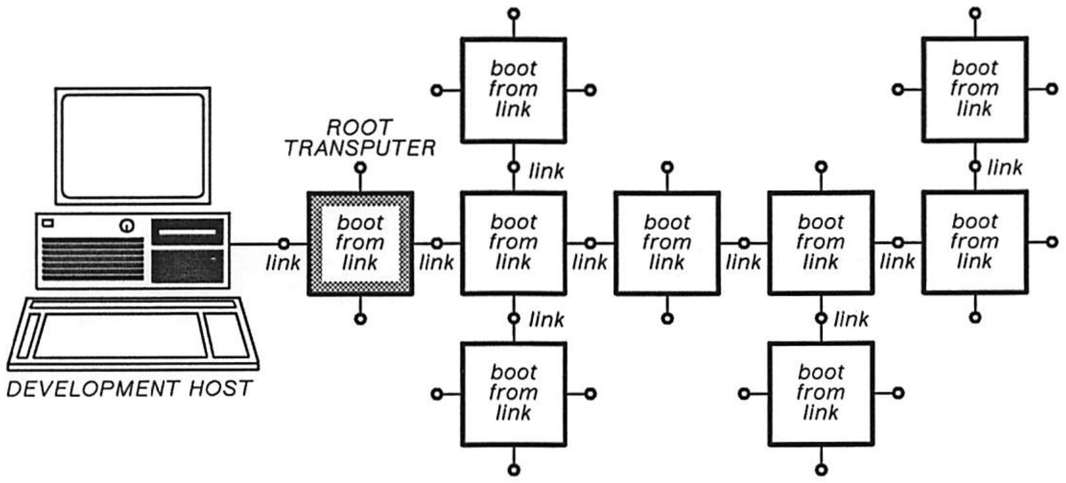
1. The code to be loaded on several processors needs to be sent from the host only once; intermediate transputers running the distributing loader may send the code out via more than one link, thus duplicating the code.

2. The distributing loader does not reside permanently in the target processor's memory.

3. The loader is small enough to fit the on–chip RAM, so that processors with vast external memory can be loaded via processors without external memory.

4. All communications are from the host to the network. This permits to boot the network by running a single program on the host, that would read a single sequential file, containing the bootstrap code, application code, routing and loading info.

Network loading algorithm – typical topology:



TRANSPUTER NETWORKS: ERROR ANALYSIS

The debugging process requires that the state of each processor is available to the programmer, should an error occur.

PROBLEM: How is it possible to retrieve this state, if it requires interaction with some program? Loading this program for the post–mortem analysis amounts to overwriting (thus losing) some memory and register information...

ANSWER: Before loading the analysing program, "peek" selected registers and memory locations and copy their contents to another processor or host. Only then load the analyser.

Transputer analysis hardware support features:

1. If a transputer runs software with `HaltOnError` *flag set, an error (subscript, etc.) causes the processor to halt, and it is possible to "peek" the address of the instruction causing error.*

2. The Analyse signal causes other transputers to stop, preserve their internal states, and boot, without damaging the preserved info. In particular, it is possible to retrieve the process descriptor and the instruction pointer of the (then) current process.

3. The Analyse signal causes the transputer to abandon initialization of the external memory (if any), thus preserving its contents.

TRANSPUTER NETWORKS: ERROR ANALYSIS SEQUENCE

The debugging sequence is similar to the booting and loading sequence, provided that before booting each processor we "peek" a part of its memory and copy its contents to the host. The steps:

1. *The host sends the bootstrap code to the root transputer, which starts to execute it.*
2. *The bootstrap program:*
   a. *saves register values from the previous program,*
   b. *initializes the transputer,*
   c. *loads and starts the distributing loader.*
3. *The loader program:*
   a. *transmits info saved by bootstrap to the host,*
   b. *initializes the transputer,*
   c. *loads and starts the analyser program.*
4. *The analyser:*
   a. *"peeks" the neighbouring transputers not yet booted,*
   b. *boots them by sending bootstrap packets,*
   c. *distributes code and info to other processors,*
   d. *conveys info from other processors to the host.*
5. *The host starts the debugger program, which interacts with the analyser programs running on individual processors.*

TRANSPUTER NETWORKS: ISSUES IN DEBUGGING OF PARALLEL PROGRAMS

A conventional, sequential program has a single thread of execution, single stack and heap. A parallel program can be considered a set of cooperating sequential programs, each with its own code, history, stack, heap, etc...

Special issues of:

Deadlock: *involves a collection of processes unable to proceed, mutually waiting for one another to do something.*

Livelock: *involves a collection of processes repetitively interacting with one another, but never communicating with the outside world.*

Practical problems:

1. *Lack of global environment, which in the conventional, sequential language would permit the programmer to introduce some print statements into the code to facilitate tracking the execution of the program. In parallel programming, a process running on a transputer in the middle of the network may have no access to the host's screen.*

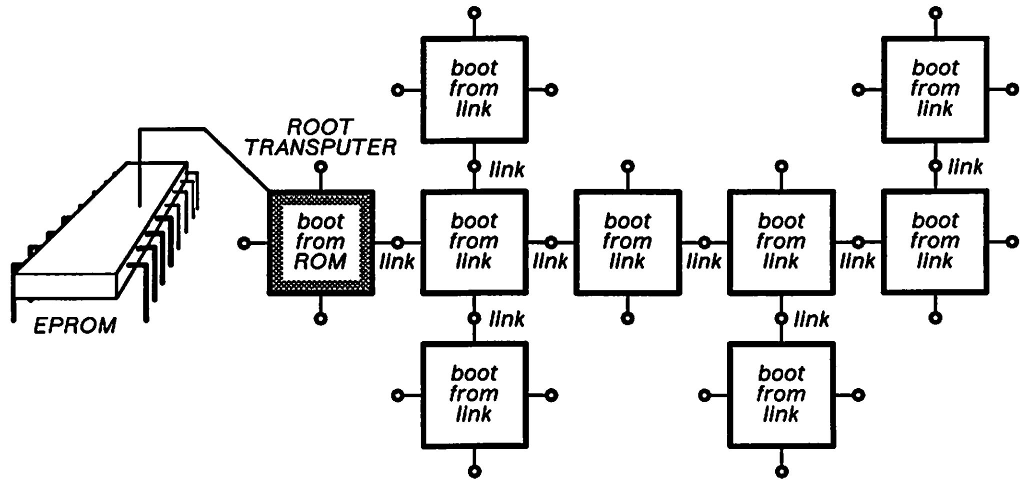
2. *In distributed systems the notions of breakpointing and single–step execution lose their usefulness.*

TRANSPUTER NETWORKS: TARGET SYSTEM INSTALLATION

Two issues need to be addressed:

1. *how to autoboot an application on a standalone network?*
2. *how to explore the topology of a standalone network?*

Booting from ROM – typical topology:



NOTES:

1. *Each transputer has a* `BootFromROM` *pin. When set at startup time, it makes the processor jump to execute code two bytes below of the top of its memory (i.e. at the beginning of the distributing loader program).*

2. *INMOS supplies a ROM loader program.*

3. *The application can and should be committed to ROM at a late stage of its development.*

TRANSPUTER NETWORKS:
EXPLORING UNKNOWN TOPOLOGIES

Exploration is useful in confirming that a a network has a particular topology, and that all processors work properly.

The worm program:

1. *Reset the network,*

2. *The host boots the root processor with the worm,*

3. *The worm tests the links of its processor to see if they are connected to other processors (using timeouts):*

   a. *if so then the check is made whether they are already booted (by "peeking" their memory),*
   b. *if links not connected or all processors are booted, then the worm reports its findings to the host,*

   else

   a. *the worm loads a copy of itself to its workspace,*
   b. *the worm boots all unbooted neighbouring processors with the copy of itself,*
   c. *the worm reports its findings to the host.*

REFERENCES AND RECOMMENDED READING

INMOS Ltd.: OCCAM 2 Reference Manual, *Prentice Hall, 1988.*

Wayman, R.: Transputer Development Systems, *in Harp, G. (ed):* Transputer Applications, *Pitman Publishing, 1989.*

Stepney, S.: GRAIL: Graphical Representation of Activity, Interconnection and Loading, *in* Parallel Programming of Transputer Based Machines, *T. Muntean (ed), IOS, 1988.*