

## Transputer occam software development and debugging tools

### FEATURES

- occam 2.1 compiler.
- Excellent compile time diagnostics.
- Some global and local optimizations.
- Assembler inserts.
- Support for EPROM programming.
- Support for placing code and data in user specified memory locations.
- Support for dynamically loading programs and functions.
- Small runtime overhead.
- Cross-development from PC and Sun-4 platforms.
- Support for explicit parallelism.

### INQUEST Interactive and post-mortem debugging:–

- Windowing interface using X Windows or Windows.
- Programmable command language.
- Source code or assembly code view.
- Stack trace-back facility.
- Variable and Memory display facility.
- C expression interpreter.

### INQUEST Interactive debugging:–

- Process and thread break points.
- Single stepping of threads.
- Read/Write/Access watch point capability.
- Facilities to interrupt and find threads.

### Performance analysis tools:–

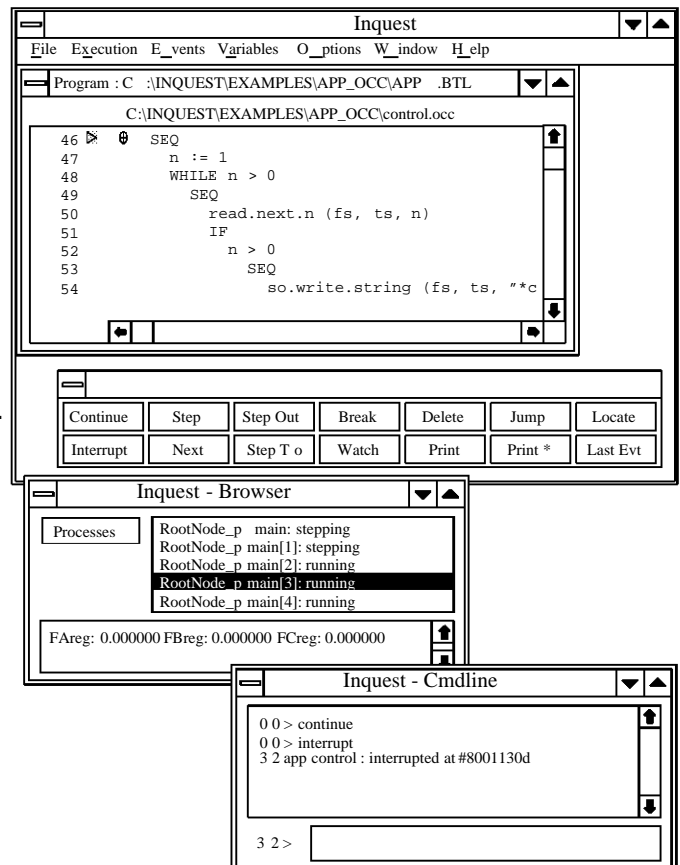
- Analysis of time spent in each function.
- Analysis of processor idle time and high priority time.
- Analysis of number of times each block executed.

### DESCRIPTION

The Transputer occam 2.1 Toolset provides a complete high quality software development environment for T2xx, T4xx and T8xx transputers and ST20450 processors. The compiler supports the extended occam 2.1 language. Improved embedded application support is provided by both configuration and symbol map utilities.

An *interactive windowing debugger* provides single stepping, breakpoints, watchpoints and many other features for debugging sequential and multi-tasking programs. *Execution profilers* give various post-mortem statistical analyses of the execution of a program.

### PRODUCT INFORMATION



# Contents

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b> .....                                 | <b>3</b>  |
|          | 1.1 Applications .....                                    | 3         |
| <b>2</b> | <b>Code-building tools</b> .....                          | <b>4</b>  |
|          | 2.1 Target systems .....                                  | 4         |
|          | 2.2 Mapping occam programs onto transputer networks ..... | 4         |
|          | 2.3 occam 2.1 development system .....                    | 8         |
| <b>3</b> | <b>INQUEST windowing debugger</b> .....                   | <b>13</b> |
|          | 3.1 Interactive debugging .....                           | 16        |
|          | 3.2 Post-mortem debugging .....                           | 17        |
| <b>4</b> | <b>Execution analysis tools</b> .....                     | <b>18</b> |
|          | 4.1 Execution profiler .....                              | 18        |
|          | 4.2 Utilization monitor .....                             | 19        |
|          | 4.3 Test coverage and block profiling tool .....          | 19        |
| <b>5</b> | <b>Host interface and AServer</b> .....                   | <b>22</b> |
|          | 5.1 The application loader – irun .....                   | 22        |
|          | 5.2 AServer .....   | 22        |
|          | 5.3 AServer features .....                                | 23        |
| <b>6</b> | <b>ANSI C Toolset product components</b> .....            | <b>24</b> |
|          | 6.1 Documentation .....                                   | 24        |
|          | 6.2 Software Tools .....                                  | 24        |
|          | 6.3 Software libraries .....                              | 24        |
| <b>7</b> | <b>Product Variants</b> .....                             | <b>26</b> |
|          | 7.1 IMS D7405 IBM PC version .....                        | 26        |
|          | 7.2 IMS D4405 Sun 4 version .....                         | 26        |
| <b>8</b> | <b>Support</b> .....                                      | <b>26</b> |
| <b>9</b> | <b>Ordering Information</b> .....                         | <b>27</b> |

# 1 Introduction

The OCCAM 2.1 Toolset provides a complete high quality software development environment for all IMS T2xx, T4xx and T8xx series transputers and ST20450 processors.

The OCCAM programming language is a high-level language which supports the design and implementation of concurrent systems on networks of processors. The OCCAM 2.1 language is described in the *OCCAM 2.1 Reference Manual*. New features of OCCAM 2.1 include:

- named types;
- record and packed record types;
- implied typing of literals;
- functions returning results of any fixed length;
- channel array constructors.

The toolset enables parallel programs to be built for single transputers or multi-transputer networks comprising one or more transputer types. Networks of processors are supported by a software through-routing configurator that greatly simplifies the placement of code and communication paths in complex applications. Improved embedded application support is provided by both configuration and symbol map utilities.

An interactive windowing debugger provides single stepping, breakpoints, watchpoints and many other features for debugging sequential and parallel programs running on one or more transputers. Execution analysis tools give post-mortem statistical analyses including execution profiles, processor utilization, test coverage and block profiles.

The host interface is provided by the AServer. This can be used simply as an application loader and host file server, invoked by the `irun` command. The INQUEST tools have their own commands which in turn load `irun` in order to load the application. The AServer may also be used to customize the host interface if required.

## 1.1 Applications

- Single processor embedded systems
- Multiprocessor embedded systems
- Massively parallel applications
- Evaluation of concurrent algorithms
- Low cost single chip applications
- Low level device control applications
- Scientific programming

## 2 Code-building tools

The occam 2.1 Toolset provides complete OCCAM cross-development systems for transputer targets. They can be used to build parallel programs for single transputers and for multi-transputer networks consisting of arbitrary mixtures of T2xx, T4xx and T8xx transputer types. Programs developed with the toolset are both source and binary compatible across all host development machines.

The occam 2.1 Toolset is available for the following development platforms:

- IBM 386/486 PC and compatibles under MSDOS 5 and Windows 3.1, or later versions.
- Sun 4 under SunOS 4.1.3 or Solaris 2.4 with X11 Release 4 server or OpenWindows 3, or later versions.

### 2.1 Target systems

The compiler will generate code for the full range of first generation transputers (IMS M212, T212, T222, T225, T400, T414, T425, T800, T801, and T805) and the ST20450.

The processor target type and other compilation options are specified by command line switches. Libraries may contain code compiled for several different target processors; the linker will select the correct target at link time. The compiler, linker and librarian additionally support code compiled to run on a range of processor types achieving a space saving in libraries.

Mixed networks may consist of any combination of any processor types. The configuration tools, EPROM support tools and interactive and post-mortem debugging tools all support mixed networks.

### 2.2 Mapping occam programs onto transputer networks

The OCCAM programming model consists of parallel processes communicating through channels. Channels connect pairs of processes and allow data to be passed between them. Each process can be built from a number of parallel processes, so that an entire software system can be described as a process hierarchy. Each channel has a **PROTOCOL** which determines the types of messages that may flow across it. This model is consistent with many modern software design methods.

Figure 1 shows a collection of four processes communicating over channels. The **multiplexor** process communicates with a host computer and hands out work to be done to one of three **worker** processes. Results from the workers are then returned to the host by the multiplexor. The following examples show how this collection of processes can be described in OCCAM and how the OCCAM program can be mapped onto a network of processors.

The OCCAM example in figure 2 illustrates how to program the collection of parallel processes in figure 1. It assumes that the **multiplexor** and **worker** processes have been compiled and the code has been placed in the transputer code files **mux.tco** and **worker.tco**.

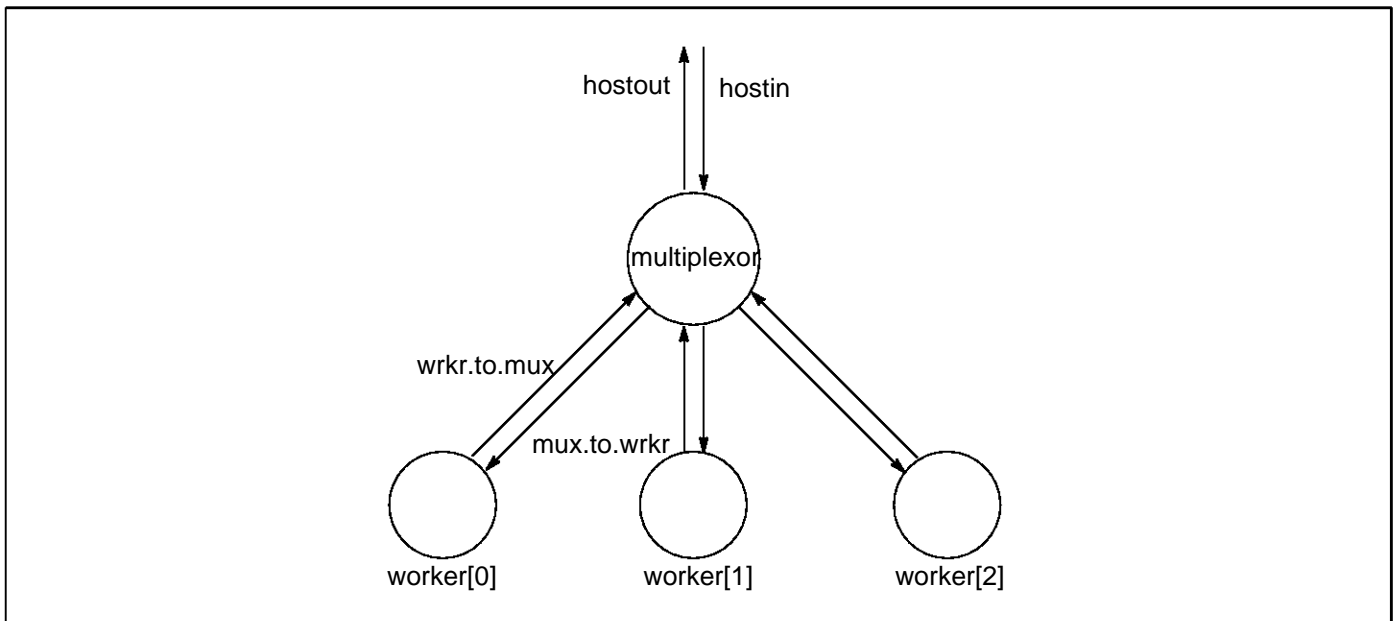


Figure 1 Software network

```

#include "hostio.inc" -- contains SP protocol definition
#USE "mux.tco"
#USE "worker.tco"
PROC example (CHAN OF SP hostin, hostout)
  [3]CHAN OF DATA wrkr.to.mux, mux.to.wrkr:
  PAR
    multiplexor(hostin, hostout, wrkr.to.mux, mux.to.wrkr)
  PAR i = 0 FOR 3
    worker(mux.to.wrkr[i], wrkr.to.mux[i])
  :

```

Figure 2 Parallel processes in occam

The **occam** compiler in the toolset can be used to compile the program shown in figure 2 and produce a code file to run on a single transputer. Alternatively it may be desirable to distribute the program over a network of processors. The program can be mapped onto a network using the configuration tools. A *configuration language* is used to describe the transputer network, and the mapping of the **occam** program onto the transputer network.

The following examples illustrate how to configure this program for a transputer network. In all cases we assume the **multiplexor** and **worker** processes have been compiled and linked into the files **mux.lku** and **worker.lku** respectively. Three hardware networks are considered – the single processor network shown in figure 3 and the four-processor networks shown in figures 5 and 7.

Figure 4 shows the configuration text for mapping the software network in figure 1 onto the hardware shown in figure 3 – a single IMS T805 with 1 Mbyte of memory, connected to the host by link 0. In this example all the processes run on the root transputer. A single transputer configuration is often used to test the code before moving to a multi-processor network.

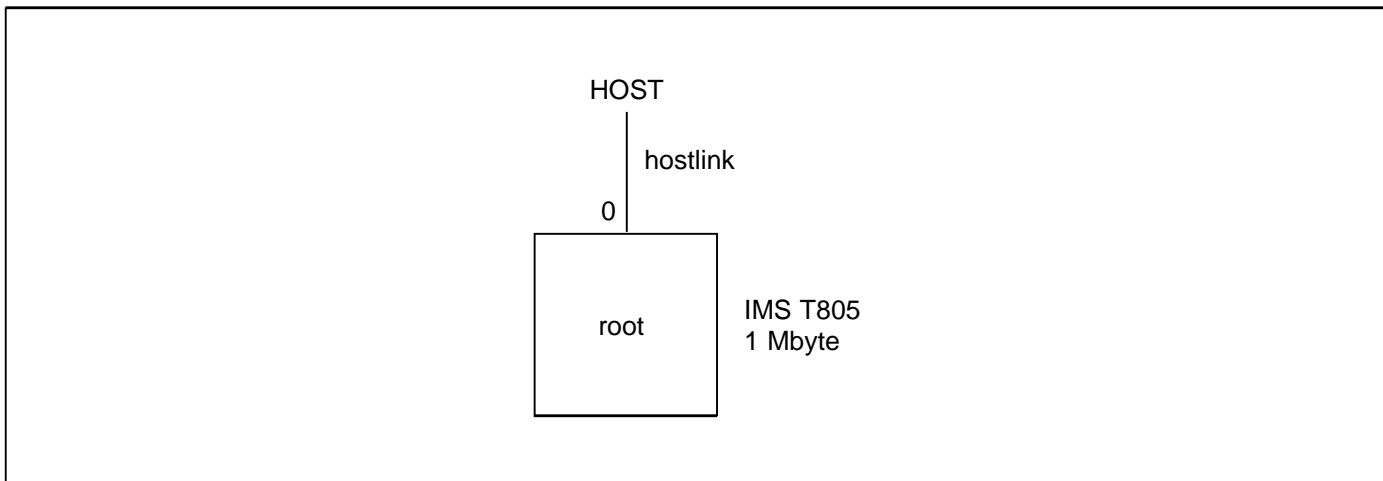


Figure 3 Single transputer network

```

VAL M IS 1024*1024:
NODE root:
ARC hostlink:
NETWORK ONE.PROCESSOR
  DO
    SET root(type, memsize := "T805", 1*M)
    CONNECT root[link][0] TO HOST WITH hostlink
  :
#INCLUDE "hostio.inc"
#USE "mux.lku"
#USE "worker.lku"
CONFIG
  [3]CHAN OF DATA mux.to.wrkr, wrkr.to.mux:
  CHAN OF SP hostin, hostout:
  PLACE hostin, hostout ON hostlink:
  PROCESSOR root
    PAR
      multiplexor(hostin, hostout, wrkr.to.mux, mux.to.wrkr)
      PAR i = 0 FOR 3
        worker(mux.to.wrkr[i], wrkr.to.mux[i])
    :
  :

```

Figure 4 Configuration text – single transputer network

Each configuration example includes a **NETWORK** description section which describes how the hardware network is connected. Since many applications can be run on identical hardware, it may be appropriate to keep this definition in a separate file and use a **#INCLUDE** statement to reference it.

The **NETWORK** description is followed by a **CONFIG** section which is virtually identical to the parallel process code shown in Figure 2. The parallel process structure has been extended with **PROCESSOR** directives indicating which processor is to run each process. The description of the software network is almost identical in all cases; only the allocation of the **worker** processes is different. If a **MAPPING** section is used, it can be arranged that the software is identical in all cases.

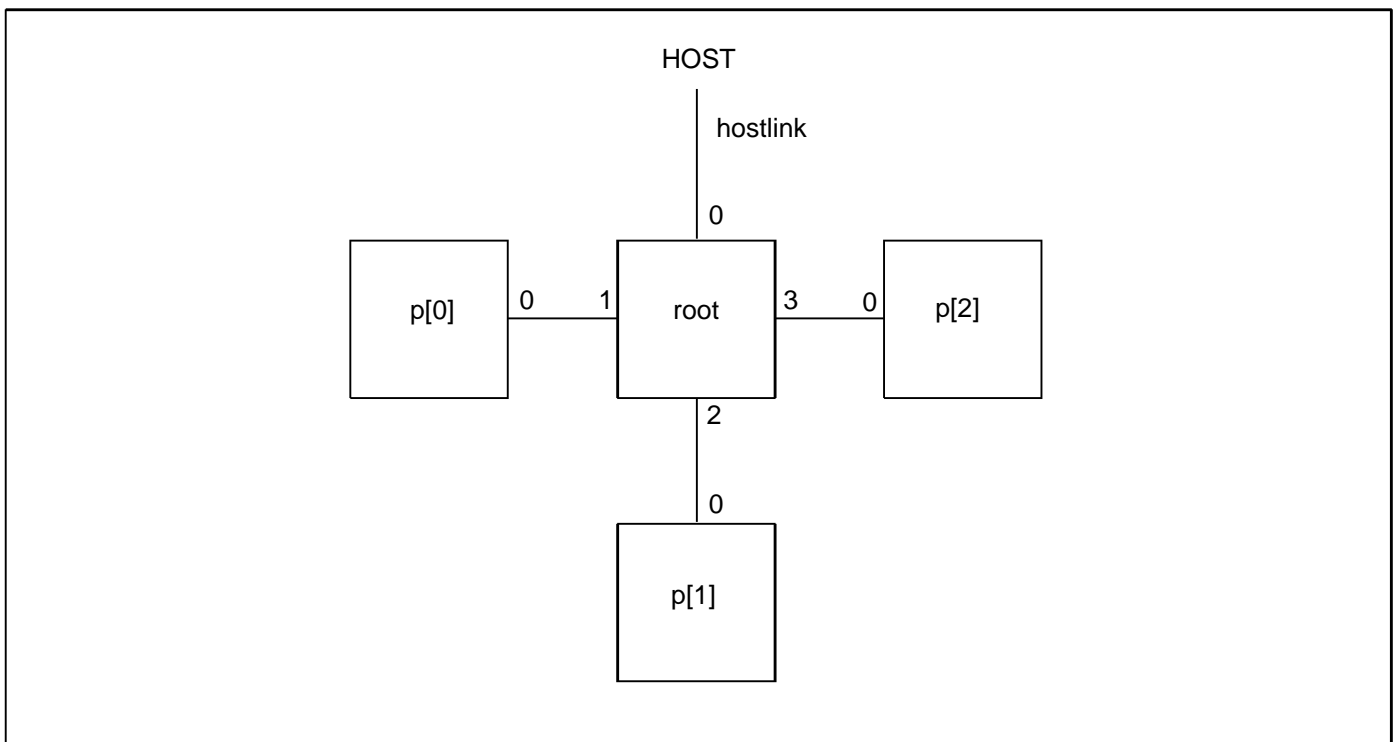


Figure 5 Four transputers arranged as a tree

```

VAL M IS 1024*1024:
NODE root:
[3]NODE p:
ARC hostlink:
NETWORK FOUR.PROCESSOR
DO
  SET root(type, memsize := "T805", 1*M)
  CONNECT root[link][0] TO HOST WITH hostlink
  DO i= 0 FOR 3
    DO
      SET p[i](type, memsize := "T805", 1*M)
      CONNECT root[link][i+1] TO p[i][link][0]
    :
  #INCLUDE "hostio.inc"
  #USE "mux.lku"
  #USE "worker.lku"
  CONFIG
  CHAN OF SP hostin, hostout:
  PLACE hostin, hostout ON hostlink:
  [3]CHAN OF DATA wrkr.to.mux, wrkr.to.mux:
  PAR
    PROCESSOR root
      multiplexor(hostin, hostout, wrkr.to.mux, mux.to.wrkr)
    PAR i = 0 FOR 3
      PROCESSOR p[i]
        worker(mux.to.wrkr[i], wrkr.to.mux[i])
  :

```

Figure 6 Configuration text – four transputers as a tree

Figure 6 shows the configuration text for mapping the software network on to the multi-transputer hardware shown in figure 5 – four IMS T805s, each with 1 Mbyte of memory. In this example the **multiplexor** process runs on the root transputer while the individual **worker** processes run on each of the other transputers.

In figures 5 and 6, processes on different processors communicate with each other over direct transputer links; each channel pair is mapped onto a single link. If the network is as shown in figure 7, then the channels cannot be mapped onto the hardware so easily. In this case the configurer can automatically add communication routing processes onto the transputers so that the user processes can be mapped onto the processors as required. Figure 8 shows the configuration file for this network.

For each multiprocessor system architecture, the connection of multiprocessor and **worker** processes is the same; only the hardware description is changed.

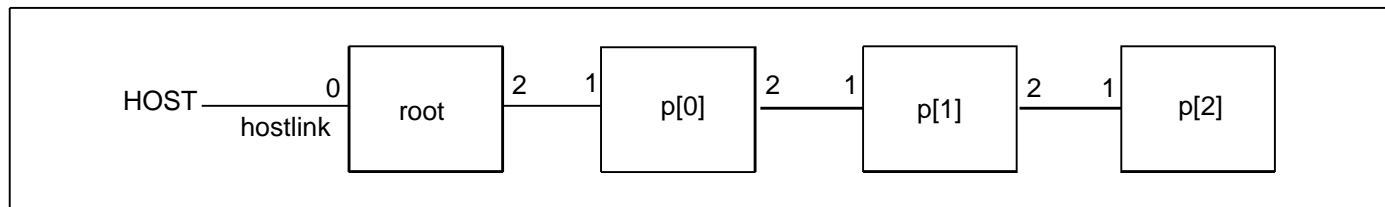


Figure 7 Four transputers arranged as a pipeline

```

VAL M IS 1024*1024:
NODE root:
[3]NODE p:
ARC hostlink:
NETWORK FOUR.PROCESSOR
DO
  SET root(type, memsize := "T805", 1*M)
  CONNECT root[link][0] TO HOST WITH hostlink
  DO i= 0 FOR 3
    SET p[i](type, memsize := "T805", 1*M)
    CONNECT p[0][link][1] TO root[link][2]
    DO i= 1 FOR 2
      CONNECT p[i][link][1] TO p[i-1][link][2]
  :
#INCLUDE "hostio.inc"
#USE "mux.lku"
#USE "worker.lku"
CONFIG
  CHAN OF SP hostin, hostout:
  PLACE hostin, hostout ON hostlink:
  [3]CHAN OF DATA mux.to.wrkr, wrkr.to.mux:
  PAR
    PROCESSOR root
      multiplexor(hostin, hostout, wrkr.to.mux, mux.to.wrkr)
    PAR i = 0 FOR 3
      PROCESSOR p[i]
        worker(mux.to.wrkr[i], wrkr.to.mux[i])
  :

```

Figure 8 Configuration text – four transputers as a pipeline

## 2.3 occam 2.1 development system

The occam 2.1 compiler supports the full OCCAM 2.1 language as defined in the *OCCAM 2.1 Reference Manual*. In addition to type checking the compiler will validate programs to ensure that variables and communication channels are being used correctly in parallel components of a program. This detects many simple programming errors at compile time. The language provides support for low-level programming, including the allocation of variables to specific memory addresses, and the inclusion of transputer assembly code.



The compiler will generate code for the full range of transputer types (IMS M212, T212, T222, T225, T400, T414, T425, T426, T800, T801, and T805) and ST20450. Since the different processor types share a common core of instructions it is possible to create compiled code which will run on a range of processors.

A program may be compiled in one of several 'error modes' which determine the behavior of the program when a run-time error occurs. A mode which supports the use of the post-mortem debugger may be chosen; in this mode the network will come to a halt when a run-time error occurs. A compiler option allows all error checking to be removed from time-critical or proven parts of a program, and unchecked routines can be called from within a checked system.

The processor target, error mode and other compilation options are specified by command line switches.

Collections of procedures and functions can be compiled separately with the `occam 2.1` compiler. Separately compiled units may be collected together into libraries. The linker is used to combine separately compiled units into a program to run as a self contained unit or process. The linker supports selective loading of library units, and the linking of components written in other programming languages including ANSI C.

The toolset supports the use of **make**, a program building tool available under UNIX and other operating systems. The input to **make** is a 'Makefile' which describes how a program is to be built from its parts, and **make** rebuilds those parts of the program which have changed. In the `occam 2.1` toolset a tool called **imakef** is provided which will generate a Makefile automatically from the `occam` program text. This guarantees that the Makefile is consistent with the program sources.

To create a multi-transputer program, the mapping of the processes to processors is described in the configuration description. This description is processed by a pair of tools called the *configurer* and the *collector*. The configurer checks the description, and passes information to the collector on how the program code and data should be mapped onto the network. The collector creates the bootstrap and routing information necessary to load the entire network, and stores this, along with the compiled code, in a program code file. The program code file can be down-loaded to one transputer in a network, and the program will automatically boot all the processors in the network and distribute the application code to them.

An application loader on the host, called **irun**, can be used to load programs on to transputer networks. Once loaded, the programs start automatically. **irun** supports access to the host terminal and file system from the transputer network. **irun** is part of the AServer, which is described in section 5.

A compiled and configured program may be run on a network under the control of the INQUEST interactive windowing debugger supplied with the `occam 2.1` toolset. The INQUEST debugger supports advanced operation including single stepping, breakpoints, watchpoints, process halting and continuation, multiple windows and command language operation. The INQUEST debugger runs on the host and therefore does not need an extra transputer in the network. If the transputer network halts because of a run-time error, or if the user interrupts the server, the debugger may be used, in post-mortem mode, to investigate the state of the network in terms of the program source text. The debugger is described in more detail in section 3.

The toolset also includes several tools for profiling applications and analyzing their behavior, which are described in section 4.

As an alternative to loading the program code from a host, the program code may be placed into an EPROM. The program code for a whole network of processors may be booted and loaded from a

single processor with a ROM. The program code file may be converted to an industry standard format for programming EPROMs, using the EPROM tools in the toolset.

A transputer network may consist of any combination of supported processor types. The configuration tools, EPROM support tools and debugging tools all support mixed networks.

### 2.3.1 Libraries

The OCCAM toolsets provide a wide range of OCCAM libraries, including mathematical functions, string operations and input/output functions. The libraries support similar functions across the full range of transputer types, making it easy to port software between transputer types. Sources of most of the libraries are provided, for adaptation if required.

The libraries provided are described below.

#### **OCCAM compiler library**

This is the basic OCCAM run-time library. It includes multiple length arithmetic functions, floating point functions, IEEE arithmetic functions, 2D block moves, bit manipulation, cyclic redundancy checks, code execution and arithmetic instructions. The compiler will automatically reference these functions if they are required.

#### **Mathematics libraries**

Single and double length mathematics functions (including trigonometric functions) are provided. These libraries use floating point arithmetic and will produce identical results on all processors. The OCCAM source code is also provided.

A separate library is provided for optimized mathematical functions for the T414, T425, T400 and ST20450 processors. These functions provide slightly different results to the maths library above but within the accuracy limits of the function specifications. The OCCAM source code is also provided.

#### **string.lib**

String manipulation procedures. The OCCAM source code is also provided.

#### **hostio.lib**

Procedures to support access to the host terminal and file system through the iserver. The OCCAM source code is also provided.

#### **streamio.lib**

Procedures to read and write using input and output streams. The OCCAM source code is also provided.

#### **crc.lib**

Procedures to calculate the cyclic redundancy check (CRC) values of strings.

#### **convert.lib**

Procedures to convert between strings and numeric values.

#### **xlink.lib**

Procedures implementing link communications which can recover after a communication failure.

## `debug.lib`

Procedures supporting the insertion of debugging messages and assertions within a program, for use with the INQUEST debugger.

### 2.3.2 Mixed language programs

It is often appropriate in the development of a large system to use a mixture of programming languages. For example, it may be necessary to preserve an investment in existing C code while using OCCAM to express the concurrency and communication within the system.

The OCCAM programming model provides an excellent vehicle for building mixed language systems where the components built in each language can be clearly defined with a simple interface. The OCCAM toolset can be used to bind these components together and distribute them over a network of transputers.

The OCCAM toolset supports calling C functions directly from OCCAM. It is possible to call C functions which require static data or heap; OCCAM procedures are provided to set up OCCAM arrays for use as static or heap areas for collections of C functions.

The associated ANSI C toolset allows OCCAM procedures and (single valued) OCCAM functions to be called from C just like other C functions. In addition OCCAM programs can be mixed with C programs at the configuration level, providing a message based interface between C and OCCAM code.

### 2.3.3 Support for embedded applications

The toolset has been designed to support the development of embedded applications. Features include facilities to:

- place code and data at particular places in memory,
- locate where functions and variables reside in memory,
- access the transputer instruction set efficiently from OCCAM.

### 2.3.4 Placing code and data

The highest level OCCAM processes in a system are visible to the configurer and identified in the configuration language; these are known as configuration processes. Each processor runs one or more configuration processes.

A configuration process consists of its code, workspace and vectorspace. The configurer allocates each of these separate segments a place in the memory of the processor.

By default, the configurer allocates all the code and data segments to a contiguous default block of memory. The location of this default block and the order of the segments may be defined in the configuration description. The configuration description can also specify that particular segments of an application are to be allocated to particular places in memory outside the default block.

The compiler, linker and collector each will optionally produce a listing of how the various parts of an application are mapped into the segments of memory. A tool is provided that can read all these map files and produce a summary of the whole application, giving the locations of all the functions, workspace areas and vectorspace areas.

### 2.3.5 Bootstraps

The source code of the standard bootstraps is provided. The user can then write bootstraps that are tailored to a specific application by using the standard ones as templates.

If the application is spread over several processors in a network, then it is possible to modify that part of the bootstrapping process which loads remote processors in order to perform special initializations on those processors before they receive any application code. If the user wishes to modify the initial or primary bootstraps, the ANSI-C toolset is required for its assembler.

### 2.3.6 Assembler inserts

The OCCAM toolset provides an assembler insert facility which supports:

- symbolic access to OCCAM variables
- pseudo-operations to load multiple values into registers
- loading the results of OCCAM expressions to registers
- labels and jumps
- directives to access particular workspace values

### 2.3.7 Dynamic loading of processes

The toolset can encapsulate the code and data of a process in a file called a *relocatable separately-compiled unit* or `rsc`. This format is suitable for dynamic loading and calling by another process. A procedure `KERNEL.RUN` and supporting functions and procedures are provided for this purpose. The `rsc` can be loaded from a file, loaded from memory, or input from a channel. The memory variant allows an `rsc` to be placed into ROM and executed if required. For example, if an application wishes to select a device driver to be placed in on-chip memory, then a number of possible drivers can be placed in ROM. The application can choose one for the occasion, copy it into low memory and execute it.

### 2.3.8 Optimized code generation

The OCCAM compiler implements a wide range of code optimization techniques:

**Constant folding.** The compiler evaluates all constant expressions at compile time.

**Workspace allocation.** Frequently used variables are placed at small offsets in workspace, thus reducing the size of the instructions needed to access them, and hence increasing the speed of execution.

**Dead-code elimination.** Code that cannot be reached during the execution of the program is removed.

**Peephole optimization.** Code sequences are selected that are the fastest for the operation.

**Constant caching.** Some constants have their load time reduced by placing them in a constant table.

**CASE statements.** The compiler can generate a number of different code sequences to cover the dense ranges within the total range.

**Inline code** Procedures and functions can optionally be implemented as inline code.

The compiler, linker and configurer provide features which allow programmers to make good use of the transputer's on-chip RAM.

### 3 INQUEST windowing debugger

The INQUEST debugger can debug OCCAM programs either interactively or post-mortem. It supports single transputer applications and multi-processor networks. A user interface displays source code or disassembled code and enables the user to interact with the debugger by means of buttons, menus, a mouse and the keyboard. The interface is built using the X Window System and OSF/Motif for Sun-4s or Microsoft Windows for PCs.

The program being debugged may consist of any number of tasks (or threads of execution) some of which may be running while others are stopped or being single stepped. The host debugger program is asynchronous and holds a copy of the last stopped state of each thread, so values may be inspected by the host while the user program is running on the network. Multiple debugging windows may be opened to view different parts of the program simultaneously.

The INQUEST debugger has three debugging modes:

- interactive debugging, i.e. monitoring the application as it executes on the target processor;
- post-mortem debugging on the target processor when the application has stopped;
- post-mortem debugging on the host from a dump file.

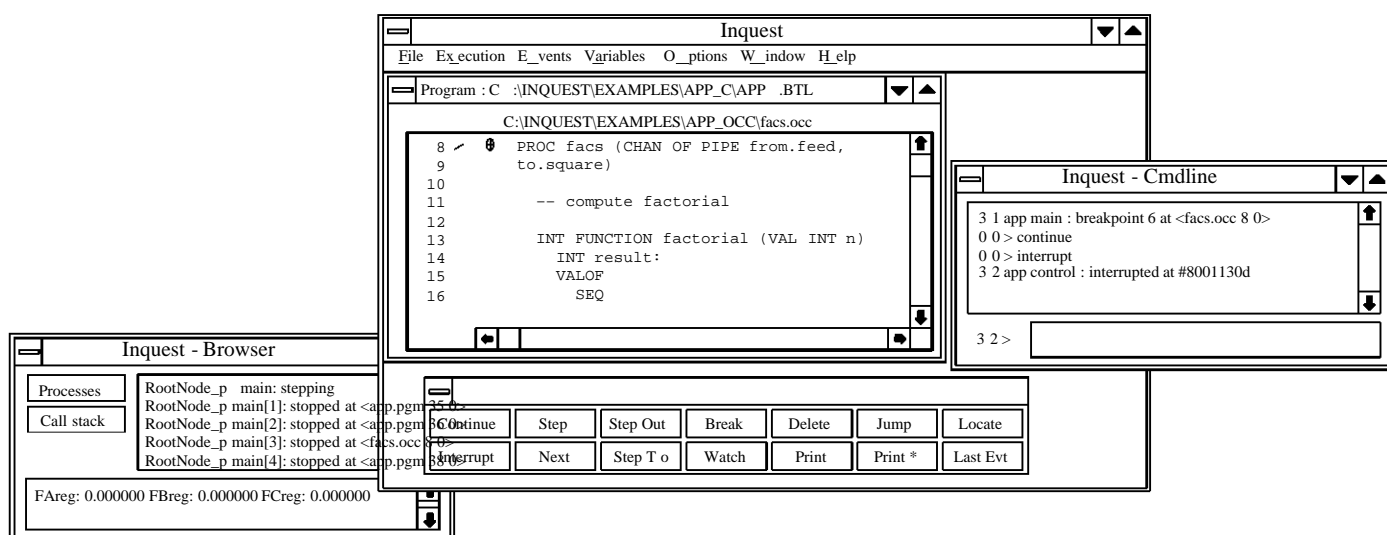


Figure 9 The Microsoft Windows debugger display

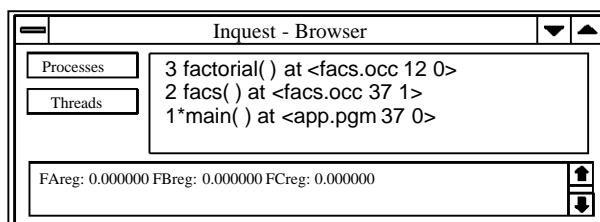


Figure 10 A Microsoft Windows stack trace

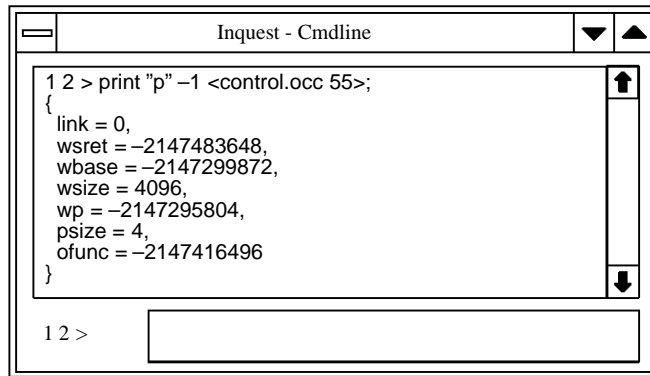


Figure 11 Displaying a structured variable on Microsoft Windows

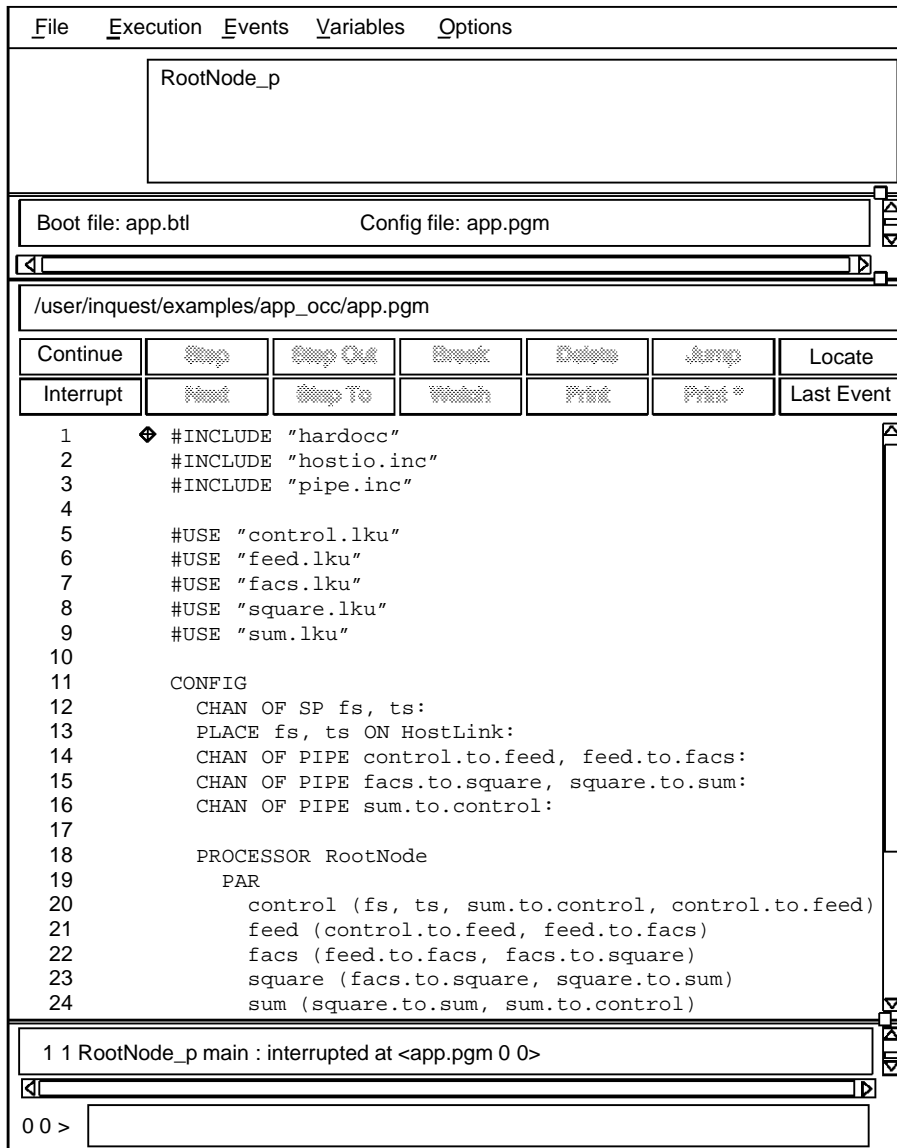


Figure 12 The X-Windows debugger display

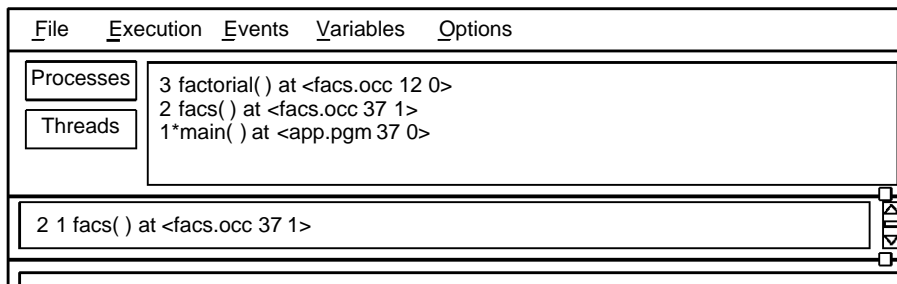


Figure 13 An X-Window stack trace

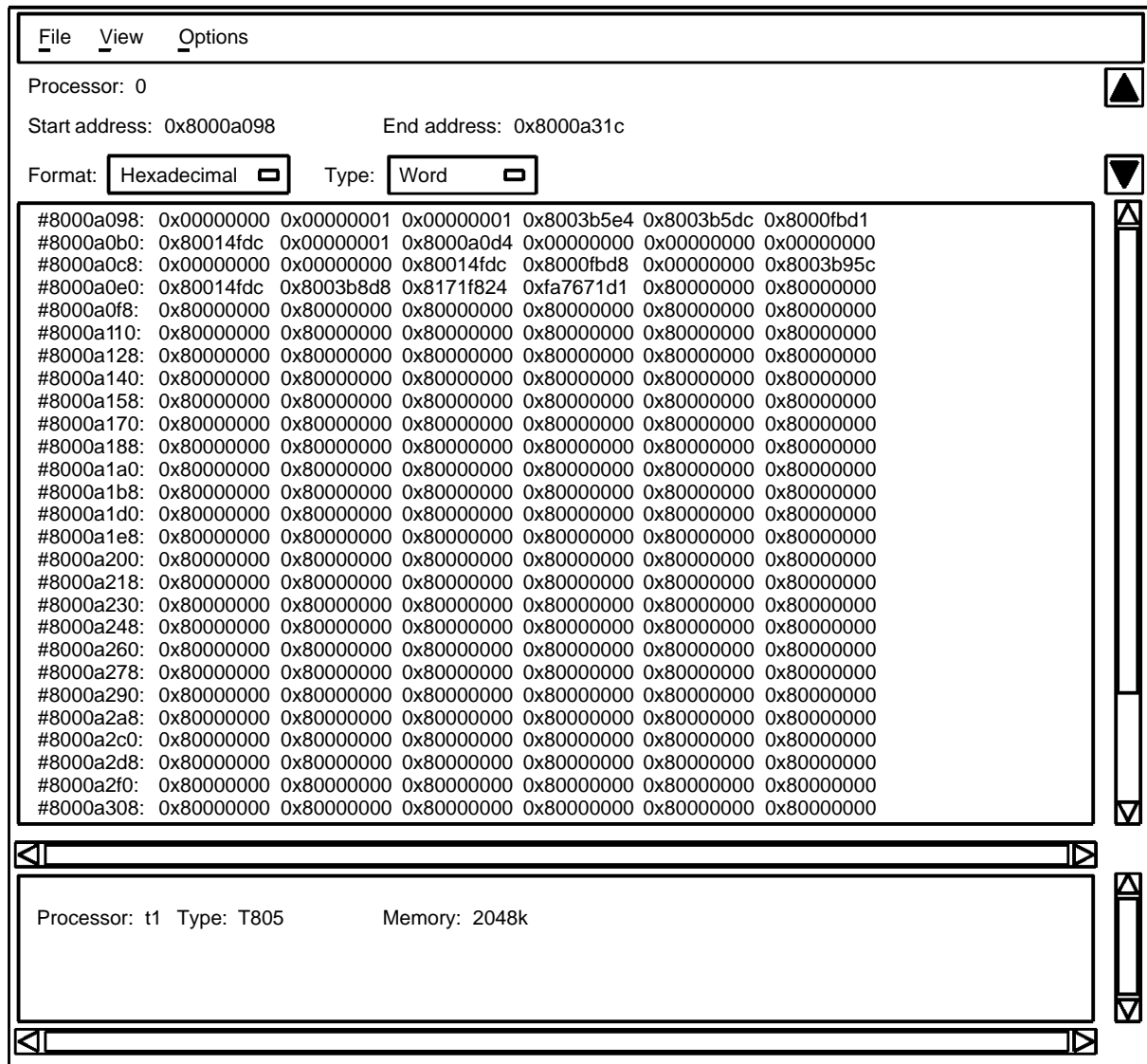


Figure 14 Displaying memory contents on X-Window

```

31      outfeed ! 0
1 2 > print "p" -1 <control.occ 55>;
{
link = 0,
wsret = -2147483648,
wbase = -2147299872,
wsize = 4096,
wp = -2147295804,
psize = 4,
ofunc = -2147416496
}
1 2 >

```

Figure 15 Displaying a structured variable on X-Windows

### 3.1 Interactive debugging

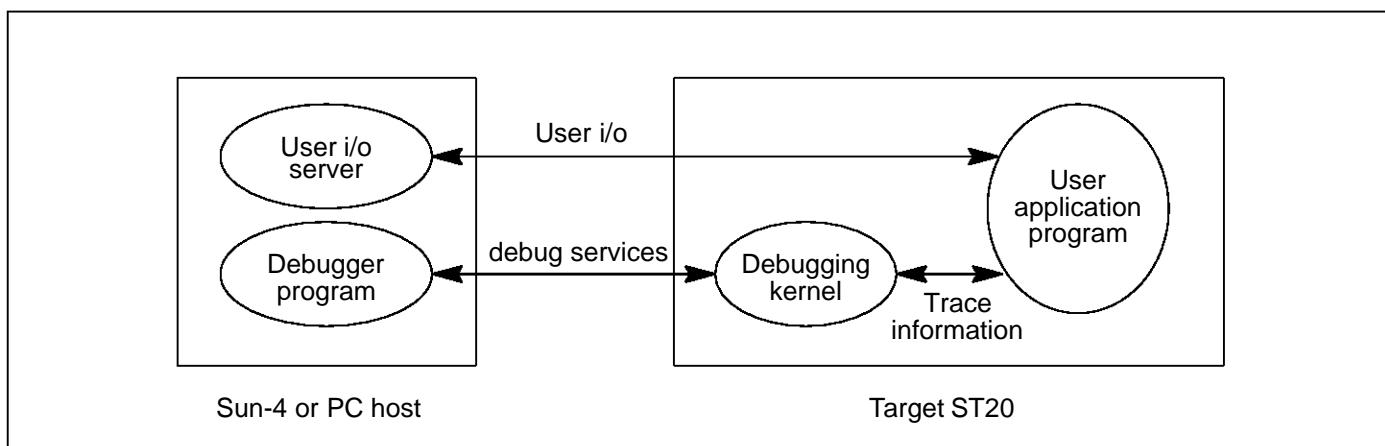


Figure 16 Interactive debugger architecture

The interactive debugger consists of a host-based symbolic debugger and a distributed debugging kernel that is configured into the application program on each transputer.

The interactive debugger provides the following features:—

- A break point facility that can be used on all or selected threads of execution.
- A single stepping facility that allows a thread of execution to be single stepped at the source level or at the assembly code level.
- A watch point facility that enables the program to be stopped when variables are to be written to or read from.
- A facility to find the threads of execution of a program and set break points on them.
- A stack trace facility.
- A facility to monitor the creation of threads of execution.
- Commands to print the values of variables and display memory.
- A simple interpreter to enable OCCAM arrays to be displayed.
- A programmable command language that allows complex break points and watch points to be set and enables debugging scripts to be generated.



- A source and object browser to select a process, a thread and source code.
- An interface to the real-time operating system for dynamic code loading and thread creation.

### 3.2 Post-mortem debugging

The post-mortem debugger provides the following features:–

- A source and object browser to select a process, a thread and source code.
- Commands to print the values of variables and display memory.
- A simple interpreter to enable `occam` arrays to be displayed.
- Creation of dump files.
- Debugging from a dump file.

## 4 Execution analysis tools

Three profiling tools are supplied for analyzing the behavior of application programs; the execution profiler, the utilization monitor, and the test coverage and block profiling tool. The execution profiler estimates the time spent in each function and procedure, the processor idle time and various other statistics. The utilization monitor displays a Gantt chart of the CPU activity of the processor as time progresses. The test coverage and block profiling tool counts how many times each block of code is executed.

The monitoring data is stored in the target processor's memory, so the profiling tools have little execution overhead on the application. After the program has completed execution, the monitoring data is extracted from the processor and is analyzed to provide displays on the program execution.

### 4.1 Execution profiler

The execution profiler uses sampling to give estimates of the total time spent executing each function on each processor.

It provides the following analyses on program execution:–

- The percentage time spent executing each low priority function.
- The percentage time spent executing at high priority.
- The percentage idle time of the processor.

The execution of the user program is monitored by a profiling kernel. The presence of the profiler kernel will slow the execution of the program by less than 5%.

The following is an example of output from the execution profiler:

```
Processor "Transptr[0]"
Idle time 51.5% (10427)
High time 0.0% (0)
Wptr Misses 0
Iptr Misses 0
Resolution 4

-----
Process "Transptr[0]_p" (100.0% processor) (9.805s)
Stack 100.0% (9805)   Heap 0.0% (0)   Static 0.0% (0)   Vector 0.0% (0)
Function Name          | Process | Processor | Samples
-----|-----|-----|-----
occama.lib/REAL64OPERR |    72.7 |    72.7   |   7133
occamutl.lib/RealIMul  |    17.5 |    17.5   |   1720
gen.occ/Generator/MatrixMul |    6.3 |    6.3   |    617
occama.lib/INT64TOREAL64 |    0.3 |    0.3   |    26
occamutl.lib/RealIDiv  |    0.2 |    0.2   |    15
convert.lib/REAL64TOSTRING/DScaleX |    0.1 |    0.1   |    7
gen.occ/Generator      |    0.1 |    0.1   |    7
convert.lib/REAL64TOSTRING |    0.1 |    0.1   |    5
convert.lib/REAL64TOSTRING/WriteDec |    0.1 |    0.1   |    5
hostio.lib/sp.write    |    0.0 |    0.0   |    4
virtual.lib/VIRTUAL.OUT |    0.0 |    0.0   |    3
hostio.lib/so.write    |    0.0 |    0.0   |    3
dblmath.lib/DRAN      |    0.0 |    0.0   |    3
convert.lib/REAL64TOSTRING/restrict |    0.0 |    0.0   |    3

virtual.lib/VIRTUAL.IN |    0.0 |    0.0   |    2
convert.lib/REAL64TOSTRING/put.byte |    0.0 |    0.0   |    2
convert.lib/REAL64TOSTRING/QuickLog |    0.0 |    0.0   |    1
gen.occ/Generator/PrintMatrix |    0.0 |    0.0   |    1
hostio.lib/so.fwrite.real64 |    0.0 |    0.0   |    1
convert.lib/REAL64TOSTRING/Round |    0.0 |    0.0   |    1
```

|                             |     |     |   |
|-----------------------------|-----|-----|---|
| hostio.lib/sp.puts          | 0.0 | 0.0 | 1 |
| occama.lib/DSCALEB          | 0.0 | 0.0 | 1 |
| gen.occ/Generator/PrintTime | 0.0 | 0.0 | 1 |

```
Processor "Transptr[1]"
Idle time 64.4% (23189)
High time 9.1% (3285)
Wptr Misses 0
Iptr Misses 0
Resolution 4
```

```
-----
Process "Transptr[1]_p" (74.3% processor) (9.515s)
Stack 100.0% (9515)   Heap 0.0% (0)   Static 0.0% (0)   Vector 0.0% (0)
Function Name          | Process | Processor | Samples
-----
occama.lib/REAL64OPERR |    72.5 |    53.9 |  6901
occamutl.lib/RealIMul  |    18.7 |    13.9 |  1779
pipe.occ/Calculator/Pipe/MatrixMul |    6.0 |    4.4 |   568
occama.lib/INT64TOREAL64 |    0.2 |    0.2 |    21
occamutl.lib/RealIDiv  |    0.2 |    0.1 |    15
pipe.occ/Calculator/Pipe |    0.0 |    0.0 |    4
```

## 4.2 Utilization monitor

The utilization monitor shows in graphical form the utilization of the processor over the time of the program execution. This is displayed by an interactive program that draws a chart of processor execution against time using X Window System and OSF/Motif on a Sun-4 or Microsoft Windows on a PC.

As with the execution profiler, the user program is monitored by a profiling kernel. The kernel will slow the program by less than 5%.

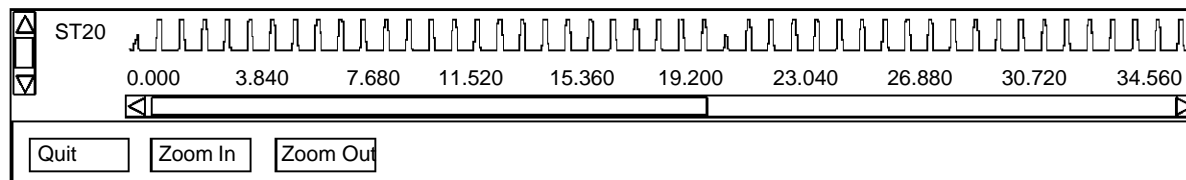


Figure 17 Example X-Windows display from the utilization monitor

## 4.3 Test coverage and block profiling tool

This tool monitors test coverage and performs block profiling for an application which has been run on target hardware.

This tool is able to:

- provide an overall test coverage report;
- provide per module test coverage reports;
- accumulate a single report from multiple test runs;
- provide a detailed basic block profiling output by creating an annotated program listing;
- provide output that can be fed back into the compiler as a part of its optimization process.

The application program (compiled with the appropriate compiler option) is run and accumulates the counts in the memory of the target processor. The tool is used to extract the results and save,

accumulate or display them. This application writes the counts into the code area, so the tool cannot be used with code running from ROM.

```

Writing coverage file "square.v" - 40% coverage
Writing coverage file "comms.v" - 14% coverage
Writing coverage file "app.v" - 75% coverage
Writing coverage file "control.v" - 36% coverage
Writing coverage file "feed.v" - 33% coverage
Writing coverage file "sum.v" - 40% coverage
Total coverage for bootable 39% over 1 run
    
```

Figure 18 Example test coverage summary report

The following is an example of the contents of a coverage file:

```

106  -- facs.ooc
106  -- generate factorials
344  #INCLUDE "pipe.inc"
0    PROC facs (CHAN OF PIPE from.feed, to.square)
106      -- compute factorial
106      INT FUNCTION factorial (VAL INT n)
344      INT result:
0      VALOF
106          SEQ
344          result := 1
0          IF
106              n > 0
344              SEQ i = 1 FOR n
0                  result := result * i
106              TRUE
344              SKIP
106          RESULT result
106      :
106      INT n:
106      BOOL going:
106      SEQ
106          going := TRUE
106      WHILE going
106          SEQ
106          from.feed ? CASE
106          data; n
106          to.square ! data; factorial(n)
106          next      -- start a new sequence
106          to.square ! next
106          end      -- terminate
106          SEQ
106          going := FALSE
106          to.square ! end
106      :
    
```

```
#####  
# Summary of results #  
#####  
Source file      : facts.occ  
Number of runs   : 1  
Processors      : All  
From linked unit : facts.lku
```

```
Line 20 - 344 times  
Line 34 - 130 times  
Line 19 - 106 times  
Line 14 - 106 times  
Line 36 - 106 times  
Line 39 - 23 times  
Line 41 - 1 time  
Line 30 - 1 time  
Line 22 - 0 times
```

```
Total number of basic blocks 9  
Basic blocks not executed    1  
Coverage 88%
```

## 5 Host interface and AServer

The host interface is provided by the AServer. This can be used simply as an application loader and host file server, invoked by the `irun` command. The INQUEST tools have their own commands which in turn load `irun` in order to load the application. The AServer may also be used to customize the host interface if required.

### 5.1 The application loader – `irun`

`irun` performs three functions, namely:

- 1 to initialize the target hardware;
- 2 to load a bootable application program onto the target hardware via the hardware serial link;
- 3 to serve the application, i.e. to respond to requests from the application program for access to host services, such as host files and terminal input and output.

These steps are normally performed when `irun` is invoked.

### 5.2 AServer

The AServer (Asynchronous Server) system is a high performance interface system which allows multiple processes on a target device to communicate via a hardware serial link with multiple processes on some external device. The AServer software acts as a standard interface which is independent of the hardware used. A simple example is shown in Figure 19, in which the external device is the host.

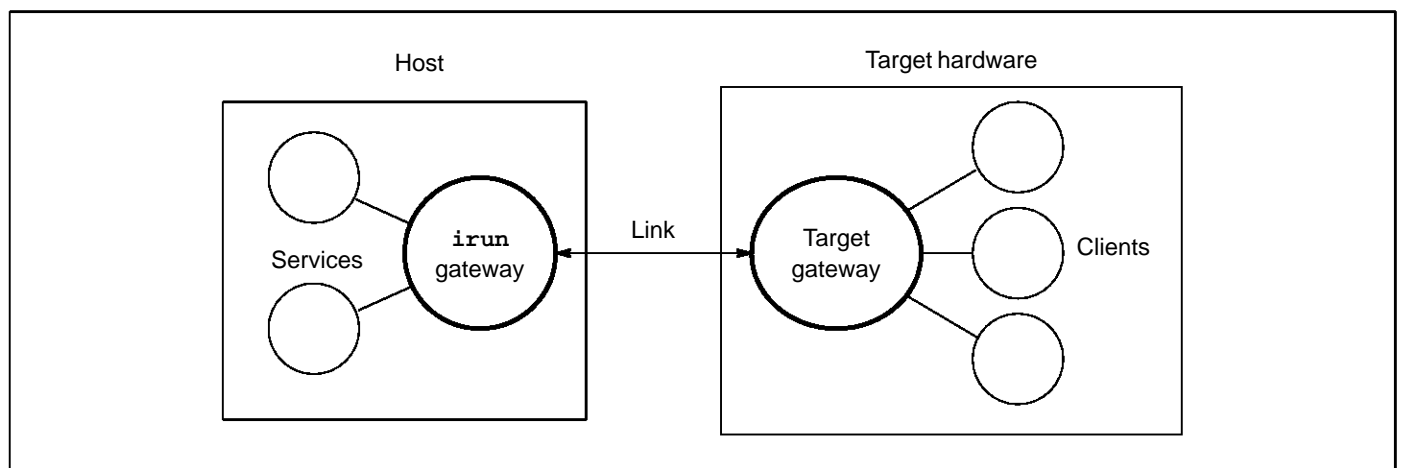


Figure 19 A simple software host-target interface

The AServer is a collection of programs, interface libraries and protocols that together create a system to enable applications running on target hardware to access external services in a way that is consistent, extensible and open. The software elements provided are:

- a target gateway which runs on the target;
- an `irun` gateway which runs on the host;
- an `iserver` service which runs on the host;
- an `iserver` converter which runs on the target;

- a library of interface routines for use by client and service processes;
- simple example services.

### 5.3 AServer features

This type of architecture offers a number of advantages:

- 1 The AServer handles multiple services.

A number of services may be available on one device, handled by a single gateway. For example, new services may be added without modifying a standard server.

- 2 The AServer handles multiple clients.

Any process on any processor in the target hardware may open a connection to any service. The process opening the connection is called the client. Several clients may access the same service. The gateway will automatically start new services as they are requested.

- 3 Services are easy to extend.

The AServer enables users to extend the set of services that are available to a user's application. The AServer provides the core technology to allow users to create new services by providing new processes. For example, the `iserver` service provides terminal text i/o, file access and system services, which may be expanded by adding new AServer service processes such as a graphics interface.

- 4 AServer communications can be fast and efficient.

The communications over the link between gateways use mega-packets, which make efficient use of the available bandwidth. Messages between the client and the service are divided into packets of up to 1 kbyte. The packets are bundled into mega-packets to send over the hardware serial link. Packets from different clients and services can be interleaved to reduce latency.

- 5 AServer communications are independent of hardware.

When an AServer connection has been established the process can send data messages of arbitrary length to the service it is connected to, receive data messages of arbitrary length and disconnect from the service. The gateways are responsible for building and dividing mega-packets and complying with hardware protocols.

## 6 ANSI C Toolset product components

### 6.1 Documentation

- `occam 2.1` toolset user guide
- Toolset reference manual
- `occam 2.1` language and libraries reference manual
- INQUEST user and reference manual
- INQUEST debugger tutorial
- AServer programmer's guide
- Delivery manual
- `occam 2.1` Reference Manual
- Tutorial introduction to OCCAM

### 6.2 Software Tools

`oc`, `ilink`, `ilibr` – OCCAM 2.1 compiler, linker and librarian

`occonf`, `icollect` – configuration tools

AServer including `irun` – host server program

`imakef`, `ilist`, `imap` – makefile generator, binary lister and memory map lister

`ieprom`, `iemit`, `imem450` – EPROM and memory interface programming tools

`inquest` debugger

`imon`, `iprof`, `iline` – execution analysis tools

`rspy` – network analyzer

### 6.3 Software libraries

- OCCAM compiler libraries
- `snglmath`, `dblmath`: mathematics functions (includes sin, cos, etc.)
- `tbmths`: mathematics functions optimized to run on T414, T425, T400 and ST20450
- `string`: string manipulation procedures
- `hostio`: file and terminal i/o procedures
- `streamio`: file and terminal stream i/o procedures
- `crc`: CRC calculation procedures
- `convert`: string-number conversion procedures



- **xlink**: extraordinary link handling procedures
- **debug**: debugging procedures.

## 7 Product Variants

### 7.1 IMS D7405 IBM PC version

All code building tools are provided in a form which will run on the host machine. Running the code, with or without interactive debugging, requires access to the transputer network. The INQUEST post-mortem debugger requires access to the transputer network unless it is using a post-mortem dump file. The profiling tools also require access to the target transputer network.

#### 7.1.1 Operating requirements

The following configuration is required:

- IBM PC with 386 or 486 processor or compatible and at least 8 Mbytes memory;
- DOS 5.0 or later and Windows 3.1 running in enhanced mode;
- 14 Mbyte of free disk space;
- A transputer development board including associated board support software.

#### 7.1.2 Distribution media

Software is distributed on 1.44 Mbytes 3.5 inch IBM format floppy disks.

### 7.2 IMS D4405 Sun 4 version

All code building tools are provided in a form which will run on the host machine. Running the code requires access to the transputer network. The INQUEST debugger requires access to the transputer network unless it is using a post-mortem dump file. The profiling tools also require access to the target transputer network.

#### 7.2.1 Operating requirements

For hosted cross-development and debugging the following configuration is required:

- A Sun 4 workstation or server with 1/4 in. tape drive capable of reading QIC-24 format;
- One of:
  - SunOS version 4.1.3, or compatible versions or
  - Solaris 2.4, or compatible versions;
- 17 Mbytes of free disk space;
- An X11 Release 4 (or later) server or OpenWindows 3;
- A transputer development board including associated board support software.

#### 7.2.2 Distribution media

Sun 4 software is distributed on DC600A data cartridges 60 Mbyte, QIC-24, tar format.

## 8 Support

TRAMs and transputer toolset development products are supported worldwide through SGS-THOMSON Sales Offices, Regional Technology Centers, and authorized distributors.


## 9 Ordering Information

| Description                   | Order number |
|-------------------------------|--------------|
| occam 2.1 Toolset for 386 PC. | IMS D7405A   |
| occam 2.1 Toolset for Sun 4.  | IMS D4405A   |

Information furnished is believed to be accurate and reliable. However, SGS-THOMSON Microelectronics assumes no responsibility for the consequences of use of such information nor for any infringement of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of SGS-THOMSON Microelectronics. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all information previously supplied. SGS-THOMSON Microelectronics products are not authorized for use as critical components in life support devices or systems without express written approval of SGS-THOMSON Microelectronics.

©1995 SGS-THOMSON Microelectronics - All Rights Reserved

IMS, occam and DS-Link are trademarks of SGS-THOMSON Microelectronics Limited.

 is a registered trademark of the SGS-THOMSON Microelectronics Group.

X Window System is a trademark of MIT.

OSF/Motif is a trademark of the Open Software Foundation, Inc.

Windows is a trademark of Microsoft Corporation.

SGS-THOMSON Microelectronics GROUP OF COMPANIES

Australia - Brazil - France - Germany - Hong Kong - Italy - Japan - Korea - Malaysia - Malta - Morocco -  
The Netherlands - Singapore - Spain - Sweden - Switzerland - Taiwan - Thailand - United Kingdom - U.S.A.