EVALUATION OF THE CD-TSE TRANSPUTER DEVELOPMENT ENVIRONMENT FOR APOLLO WORKSTATIONS

Henrik Kristensen
Evaluation of the CD-TSE transputer development environment for Apollo workstations.

Henrik Kristensen, CERN
Summer Student from Dept. of Computer Science, Univ. of Copenhagen.

Abstract:
The Cresco Data transputer development environment and remote procedure call facility for Apollo workstations, the CD-TSE, is evaluated for applications in high energy physics. The conclusion is that the product is not yet user-friendly enough for farming applications but probably soon will be. A more general product evaluation including benchmarks are performed.
1. Introduction

This document was prepared as part of a Summer Student Project at CERN august '89 and the aim is both to evaluate the CD-TSE development environment in terms of suitability for high energy physics and also to give Cresco Data feed-back in their product. This evaluation was done in collaboration with Apollo Computer.

The manufacturer of the transputer, INMOS, supplies a software development environment, TDS2, running on any IBM compatible PC. The INMOS environment consists of a special folding editor, an OCCAM compiler/linker and various other utilities.

This software is designed to run on a transputer on a board plugged into the bus of the PC and uses only the PC as terminal and fileserver.

Cresco Data has adapted this development environment to run on an Apollo Domain workstation, enhancing the user interface by utilising the Apollo Dialogue graphical user interface. This product is called the CD-TDS.

Cresco Data has also developed a product called the CD-CPS, the coprocessor server. This product enables programs running on the the Apollo workstation to make remote procedure calls and inter-process communication to programs running on the transputer system. A fileserver running on the Apollo enables the transputer programs to access the Apollo filesystem.

The development system, CD-TDS and coprocessor server, CD-CPS together are called the CD-TSE, the Transputer Software Environment.

This document evaluates the Transputer Software Environment both in terms of user friendliness and functionality and also in terms of its utilisation in high energy physics. The evaluation is not totally complete as eg. editor macros and transputer networks (>1) are not investigated. This evaluation was done with CD-TSE v.1.3.3.
2. The development system.

The CD-TDS is the Transputer Development System. It consists of a folding editor, an OCCAM compiler/linker and various other utilities. All these programs run on the transputer. It also incorporates a user interface and fileserver running on the Apollo. The user interface is implemented using the Apollo Dialogue system utilising the mouse as well as menus.

In a folding editor, whole blocks of text can be folded into one single line so only the header of the fold is visible. Text can be folded and unfolded again by simple control-keys. Since folds can be nested, text files of many thousand lines can be so well structured that they can be maintained almost without hard copy listings.

In the folding editor in the development system, folds do not have to contain text but can also contain compiled code and other binary information (Of course such folds cannot be unfolded by the user). And the development system in fact uses the fold structure as a kind of file system to store all the source, object code libraries etc.

The user has no direct access to the Apollo file system from within the development system – everything is wrapped up in a fold-hierarchy. Everything is stored by the development system in special format files in the current working directory on the Apollo, but the user has very little control of this.

All operations (compilations etc.) are performed from within the folding editor by calling a program running on the transputer with a fold as input.

2.1 The documentation of the development system.

The documentation is generally well structured except that text and tables/figures do not always come in the logical sequence.

The documentation is not a reference manual of the folding editor, OCCAM compiler etc. For this you have to refer to the reference manual for the INMOS development system. Therefore the main purpose of the documentation for the Cresco Data development system is to describe the difference between the INMOS and the Cresco Data development system. This is not done properly, mainly because table 3 on p.18, describing all the control-keys, is incomplete and erroneous.

As this development environment is so different from a conventional development environment - mainly because of the folding editor - it is almost essential to have a good tutorial. The tutorial supplied is the original INMOS development system tutorial. This is not a bad tutorial, but it would be a big improvement if the terminology in the tutorial was adapted to the terminology of the Cresco Data development system. Eg. instead of "Place the cursor on this line and press [Enter Fold]." it should say "Double-click on this line".

At this stage the documentation contains some spelling mistakes and typos. A list of those found has been forwarded to Cresco Data.

2.2 The functionality of the CD-TDS.

The CD-TDS is very reliable and user-friendly product. The fact that all common operations can be
performed with the mouse makes the system very easy to use.

The exception to this is that folds cannot be created with the mouse/menus; you have to remember that the control key to create a fold is \textquotesingle\textquotesingle N. It would be very nice if this could be done with the mouse as well. Eg. be pressing the middle mouse button on the first line of the fold and releasing it on the last line.

It is annoying that the OCCAM compiler stops at the first error it encounters and doesn't give a list of all the errors in the program. Fortunately the program can easily be modulized into SC's, separately compiled procedures, which can be nested.

This is, of cause, the responsibility of INMOS as well the fact that no run-time debugger exists. The only debugger available is just a post-mortem dump analyser. When a program is deadlocked the debugger can determine which processes are waiting on each other on which channels, but what happened before the deadlock is very hard to trace. This makes the debugging of OCCAM programs somewhat difficult and complicated.

Another inconvenience is the fact that within the development system it is not possible to change working directory. I.e. within the development system you are restricted to the working directory you were in on the Apollo when you started the development system. Moving folds can only be done with some supplied utilities, and it always involves exiting the development system and entering it again a number of times which is time consuming. It should be possible to change the working directory from within the development system enabling folds to be moved with the normal \textit{copy} and \textit{move} commands of the development system.

A list of other minor inconveniences has been forwarded to Cresco Data.

3. The coprocessor server.

The coprocessor server, CD-CPS, consists of two pieces of software running on both the Apollo and the transputer enabling interprocess communication between programs running on the transputer and programs running on the Apollo. The transputer programs are started with a remote procedure call, RPC, from the Apollo.

To use the coprocessor server you use the development system to write a transputer program as a procedure with a single entry point and compile it within the development system. You then use a special tool called the Client Stub Source Writer, CSSW, running within the development system. The stub writer takes the transputer program as input and produces a C or Pascal source as output. This source is output to an ordinary file on the workstation and must be compiled on the workstation using the workstation C or Pascal compiler. This will produce a small library with easy-to-use procedures to load the transputer program onto the transputer and call it as a procedure from within a program running on the Apollo. This procedure call can be a \textit{fork}, so the Apollo-part of your application continues to run.

You can declare a number of channels to use for interprocess communication between the program running on the Apollo and the program running on the transputer. From the transputer program these channels are plain OCCAM channels and on the Apollo side a number library functions exists to send and receive data on the channels.

The coprocessor server also includes a fileserver conforming to the AF-server protocol defined by INMOS enabling the transputer program to access the entire Apollo filesystem.
3.1 The documentation of the coprocessor server.

The documentation of the coprocessor server consist of four chapters: Overall description of the concept of the coprocessor server, command reference of the servers running on the Apollo, description of the stub writer, CSSW and finally a reference of the library providing all the facilities.

The overall description of the coprocessor server and of the stub writer are very good and thorough. After reading these chapters there should be no doubt about what the system does and how it does it.

The server command reference is very simple as there are only three commands: to start the server, to shut it down and to display current status of the server running. This status command will give you a lot of information, but how to interpret this information is not explained by a single word. This renders this command more or less useless (at least to a non-expert in transputer programming who may not be able to guess the meaning of the output).

The library reference is definitely the weakest point in the documentation as it is not a reference.

For all the library calls it describes what they do. But it is not described how to call them or more precisely: what are the types of the parameters. To find out the type of the parameters you have to find an example and guess. The problem is magnified by the unusual procedure calling conventions on the Apollo and the obsolete C compiler which doesn’t do any type-checking.

Also a list of possible error conditions and their probable causes in response to the various library calls is missing.

A general criticism of the documentation (both development system and coprocessor server) is that it assumes everybody is using AEGIS. So the UNIX user will have to think in AEGIS terms.

3.2 The functionality of the coprocessor server.

Calling OCCAM procedures on the transputer from a program on the Apollo works very well.

The system seems robust; only on two occasions has the system crashed and both reproducible. Whether the bug is the coprocessor server or in the Apollo C compiler is not clear but examples has been forwarded to Cresco Data for further investigation.

The coprocessor server can download both network configurations and code for whole networks of transputers so in principle there is no limitation on the complexity of your applications. But as you can only call one OCCAM procedure at a time, advanced applications are forced to have a non-trivial interface to the Apollo with one procedure acting as a server/multiplexer for all the "real" procedures which are probably spread across a network of transputers. But without introducing a small kernel running on the transputers this can probably not be avoided.

The interprocess communication between the Apollo program and the transputer program works fine from the transputer side, where the channels to the Apollo program are just ordinary OCCAM channels. The channels can only use a limited number of protocols, namely simple types and sized and unsized arrays of simple types. But that is perfectly adequate for almost any application.

But on the Apollo side the channels are just streams of bytes and the protocols must be implement-
ed by hand by the programmer. This is very unfortunate as it is an unnecessary source of bugs. Not conforming to the protocol will almost certainly lead to a dead-lock.

As there are only a limited number of different protocols allowed it would be very easy to implement a library of protocol handlers for the possible protocols and then let the stub writer bind these handlers to the channels giving a set of stubs to send and receive data implicitly using the correct protocol on the different channels.

3.3 Benchmarks of the coprocessor server.

To investigate the performance in terms of different Apollo/transputer transfer rates in the coprocessor server a number of benchmarks were performed.

These benchmarks were performed on an Apollo DN3500 with 4 Mb of memory and the 170 Mb disk, fitted with an INMOS B004 motherboard equipped with two 20 MHz T800 TRAMS each with 2 Mb of memory.

The INMOS board was modified to use interrupt number 7 and no DMA was used.

3.3.1 Loading and calling procedures.

To measure the time it takes to load programs and the overhead in the call of programs (in the form of, separately compiled procedures, SC’s) on the transputer, 3 dummy programs of different sizes were used. All 3 programs do nothing at all; they don’t take any parameters and return immediately without any result value. The different sizes of the programs is obtained by having a different number of non-used dummy procedures inside the main SC. The sizes of the programs (the actual .csc file) are 89, 24038 and 40695 bytes.

<table>
<thead>
<tr>
<th>Size of SC (bytes):</th>
<th>89</th>
<th>24038</th>
<th>40695</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (in ms) to call SC incl. load and unload:</td>
<td>24.4</td>
<td>234.3</td>
<td>368.1</td>
</tr>
<tr>
<td>Time to preload the SC:</td>
<td>16.13</td>
<td>223.9</td>
<td>353.6</td>
</tr>
<tr>
<td>Time to call the preloaded SC:</td>
<td>7.613</td>
<td>9.992</td>
<td>11.46</td>
</tr>
</tbody>
</table>

The conclusions made from the figures are the following:

1. The time to make the actual call is in the order of 10 ms. Slightly dependent on the size of the code in transputer. This is because the code of the SC is moved in the transputer memory before it is executed (at some astounding 16 Mb/s).

2. If an SC is to be called more than once it should definitely be preloaded if the necessary memory is available on the transputer.

3. The time to (pre-) load a program is approximately 16 ms + 9 ms/KB of code. This implies a code transfer rate of approx. 117 kb/s.
3.3.2 Transfer of parameters.

To measure the time to transfer parameters in the procedure call another set of dummy programs were used. Programs with no code but one or two parameters - arrays of bytes in different sizes.

<table>
<thead>
<tr>
<th>Array size</th>
<th>Time for VAL parameter</th>
<th>Time for two-way parameter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>10.77</td>
<td>Not measured</td>
</tr>
<tr>
<td>2 x 1 byte</td>
<td>11.43</td>
<td>14.27</td>
</tr>
<tr>
<td>1 kbyte</td>
<td>35.37</td>
<td>66.96</td>
</tr>
<tr>
<td>2 x 1 kbyte</td>
<td>60.60</td>
<td>118.79</td>
</tr>
<tr>
<td>1 Mbyte</td>
<td>26500</td>
<td>Not measured</td>
</tr>
</tbody>
</table>

This shows that the time to transfer parameters to the SC is about 3 ms + 38 kb/s of data. (Plus the 7-10 ms to make the call.) If the parameters are two-way (copy-restore parameter passing) the time to copy back is 5 ms + 38 kb/s of data.
3.3.3 Transfer of data from transputer to Apollo on a channel.

To measure the sustained transfer rate on interprocess communication channels a third set of almost dummy programs were used. They just send and receive dummy data on a channel. As with the other measurements the transfer rate was measured as a function of the block- (protocol-) size.

<table>
<thead>
<tr>
<th>Blocksize:</th>
<th>Transputer -&gt; Apollo</th>
<th>Apollo -&gt; Transputer</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Transputer (ms)</td>
<td>Apollo (ms)</td>
</tr>
<tr>
<td>5 bytes</td>
<td>2.79</td>
<td>1.75</td>
</tr>
<tr>
<td>20 bytes</td>
<td>2.88</td>
<td>6.78</td>
</tr>
<tr>
<td>68 bytes</td>
<td>3.07</td>
<td>21.6</td>
</tr>
<tr>
<td>260 bytes</td>
<td>3.88</td>
<td>65.4</td>
</tr>
<tr>
<td>1 kb</td>
<td>9.68</td>
<td>103</td>
</tr>
<tr>
<td>2 kb</td>
<td>16.6</td>
<td>120</td>
</tr>
<tr>
<td>4 kb</td>
<td>30.5</td>
<td>131</td>
</tr>
<tr>
<td>16 kb</td>
<td>114</td>
<td>140</td>
</tr>
<tr>
<td>64 kb</td>
<td>452</td>
<td>142</td>
</tr>
<tr>
<td>1 Mb</td>
<td>7211</td>
<td>142</td>
</tr>
</tbody>
</table>

It is clear that the transfer rate depends highly on the blocksize.

The reason why the transfer rate is higher to the transputer than from the transputer is probably because buffering takes place in the Apollo where the interface is asynchronous as opposed to the synchronous interface in the transputer.

It is believed that the overhead of the small blocks is mainly on the Apollo side and is not wasting cpu time in the transputer. But this remains to be verified.

3.3.4 Performance of the fileserver.

To measure the sustained performance of the file server two programs were written to read and write dummy data in a file on the Apollo. The AF-server protocol used by the fileserver unfortunately limits the record size to 512 bytes.

<table>
<thead>
<tr>
<th>Writing:</th>
<th>Reading:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Transputer (ms)</td>
</tr>
<tr>
<td>1 byte</td>
<td>12.5</td>
</tr>
<tr>
<td>32 bytes</td>
<td>13.0</td>
</tr>
<tr>
<td>64 bytes</td>
<td>13.3</td>
</tr>
<tr>
<td>128 bytes</td>
<td>14.2</td>
</tr>
<tr>
<td>256 bytes</td>
<td>16.0</td>
</tr>
<tr>
<td>512 bytes</td>
<td>18.8</td>
</tr>
</tbody>
</table>

These figures show that the transfer rate to/from a file is only about one fourth of the interprocess communication transfer rate. This degraded performance is imposed by the AF-server itself and is not the cause of physical disk access. This can be concluded by the fact that reading a file is slightly faster than writing it and that this difference is caused by the file already being in the Apollo's memory.
So if high transputer/disk performance is needed it is necessary to code your own fileserver as part of the program running on the Apollo and use large blocks (<=16kb).


The two types of applications of transputer systems are most likely in high energy physics: batch-type farming and fast real time acquisition/triggering.

In case of a real time application this will probably be written mostly in OCCAM and utilise fine grained parallelism. This would not be written by an off-the-shelf physicist but more likely by a computer scientist or electrical engineer with a solid background in parallel computing. The CD-TSE would be ideally suited for this kind of application; both using the Cresco Data development system and the Apollo workstation as user interface / back end utilising the facilities of the coprocessor server.

But in the case of a farming application there are a number of reasons why the CD-TSE at its present stage is not suitable.

OCCAM as a programming language is just a bit higher level that assembler and therefore not suitable for writing number-crunching programs. A higher level language like C or FORTRAN is needed.

This language must also have a very good debugger. And because of the lack of a debugger for OCCAM it is imperative the the physicist doesn’t have to write a single OCCAM statement in her application.

Writing parallel programs is non-trivial. Proper understanding of the principles of parallel programming is necessary just to have a 'chance' of avoiding the hazards of termination, dead-lock, starvation etc. This cannot be expected from even the best FORTRAN-programming physicist. An environment has to be provided to put these problems a bit in the background, though they can probably never be totally avoided. At the same time this environment should help distributing the workers across the transputer network and join together all the loose strings in the host transputer.

All interprocess communication on the transputers obey some user-defined protocols where only data of a certain type can be sent over a given channel quite analogous to parameters in procedure calls. This strong typing clarifies the semantics of the program and therefore improves the correctness of the program. But on the channels from the transputer to the Apollo these protocols are not enforced. On the Apollo these channels are just streams of bytes and the programmer himself has to implement the protocol. This is not only a source of error but it also forces the programmer to have detailed knowledge of the data representation on the transputer which should be totally unnecessary.

These three problems is relatively easily solvable. As you read this the next release of the CD-TSE providing C, Pascal and FORTRAN for the transputer will probably be available. And if Cresco Data doesn’t develop a general purpose farming harness for the transputers somebody else probably will. The protocol problem is best solved by enhancing the stub writer a bit.

So all in all: The future for the CD-TSE looks bright but we are not all the way yet.
Appendix A: How to modify an INMOS transputer board for the Apollo.

The INMOS board can use either interrupt request line no. 3 or no. 5. But as these two interrupts normally are used by the Apollo itself (by respectively the domain token ring network controller and the disk/streamer tape controller) it is necessary to modify the board to be able to use another interrupt number.

This is the instruction to modify the board to use interrupt no. 7 instead of number 3; thus making the board switchable between 7 and 5 instead of 3 and 5.

1. Interrupt line no. 3 goes from pin 22 of IC7 to pin 25 on the PC bus edge connector. Cut this track.

2. Solder wire from pin 22 of IC7 to pin 21 of PC connector.

To locate pin 21/25 on PC connector: On solder side of board, count to 21/25 from mounting bracket.

To use interrupt no. 5: Set switch SW1-3 to OFF.
To use interrupt no. 7: Set switch SW1-3 to ON.
Appendix B: Moving TDS-files from a PC to the Apollo.

The INMOS TDS2 and the CD-TDS use exactly the same file formats. These formats are non-ASCII so there is no problems with <CR><LF> versus <LF> as normally when transferring files between UNIX and DOS systems. So files can just be copied from the PC to the Apollo.

But as the names of the files are hidden in the fold structure, the safe way to do this is to transfer ALL files in a PC directory to an empty directory on the Apollo. A step by step instruction of how to use the Apollo PC emulator to do this is given below.

Copy the whole directory will also transfer the "toplevel.tkt" toolkit file. This file does not contain the toolkit itself but only references to the actual program files in the installation of the tds. Therefore the PC version of this file cannot be used but must be substituted by the Apollo version. This can be taken from any other directory on the Apollo. A virgin copy of the toolkit file should exist in the "/cdtds/system" directory.

Using PC-format floppy disks on the Apollo Workstation.

To read and write PC-format floppy disks you need the DPCE, Domain PC Emulator. We will assume that this piece of software is already installed and configured correctly on the machine.

The PC emulator is started up by the command /com/dpce.

Do not put your floppy in the drive before the PC emulator has loaded and given you the "C>" prompt. Otherwise it will try to boot MS-DOS from your floppy.

When the PC emulator has been loaded you should issue the command PCI (in the PC emulator). This will map the Domain filesystem onto logical drive "E:" of the "PC".

You can now copy files from a floppy to the Apollo by eg.

    copy a:\sourcedir\*.\* e:\nodename\targetdir

which will copy all files from the directory sourcedir on the floppy into the directory targetdir on the node nodename. targetdir must already exist.

Use backslash instead of forward slash; it will be translated in the Domain filesystem.

And use only single and not double backslash in front of the nodename. The nodename must be specified.

When you have finished copying click on the "EXIT" box in the PC emulator window. This will close down the PC emulator.