Cluster Interconnect Traffic Analysis

Brian Coghlan and Michael Manzke
Computer Architecture Group
Trinity College Dublin, Ireland
{Brian.Coghlan, Michael.Manzke}@cs.tcd.ie

Abstract **-- Trinity College Dublin has designed and is currently prototyping a trace instrument that allows deep traces of high speed interconnect traffic [5]. An initial implementation for the Scalable Coherent Interface (SCI) proves the concept. SCI is one of the enabling interconnect technologies for high performance computing on PC Clusters.**

Such an instrument is essential for a detailed spatial and temporal analysis of parallel executed algorithms on loosely coupled clusters. Currently, there are no commercial instruments available that sample and store very deep (>> 10Mbyte) interconnect traces per target node.

The technology enables the non-intrusive real-time acquisition of high speed interconnect traffic into a database. The database, which over time is expected to represent the major investment, provides a powerful means for a fine-grained analysis of a large quantity of trace data. This paper describes the technical features of the SCI trace instrument and outlines the tool's potential for further research and development activities.

Keywords **-- Interconnect Traffic, Trace Analysis, Trace Database**

I. INTRODUCTION

This paper describes an instrument for acquisition and analyses of interconnect traffic for clusters. The instrument provides hardware designers and software developers with a tool that allows a deeper understanding of the temporal behaviour of their hardware and software on any given target system. Unlike other systems, e.g. see [6], this trace instrument is targeted to commercially available interconnect hardware and therefore provides the user with information about the true temporal behaviour of clusters made up of standard components.

This work has been supported by ESRIT project 25257 SCI Europe

Figure 1. shows how the trace instrument's hardware and software components are related to each other during trace data acquisition and a subsequent off-line data analysis.

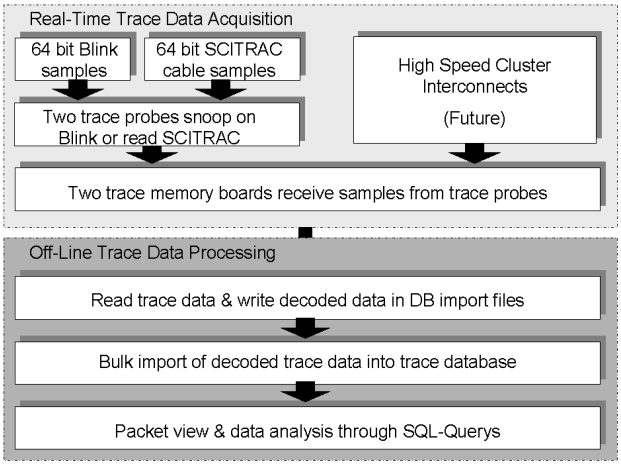

Fig. 1. Trace data flow overview

The instrument is designed to fulfil the following main objectives:

- Non-intrusive monitoring of interconnect traffic
- Very deep (>> 10Mbyte) interconnect traces per node
- Acquisition of all the interconnect traffic
- Synchronous trace acquisition on multiple nodes through a shared trigger mechanism
- Straightforward adaptation to other interconnects
- Trace data storage in commercial relational database
- Ability to analyse causal relationships in synchronously acquired traces from different targets

The utilisation of a relational database provides the user with an easy means to extend and to adapt the predefined database queries to their specific needs.

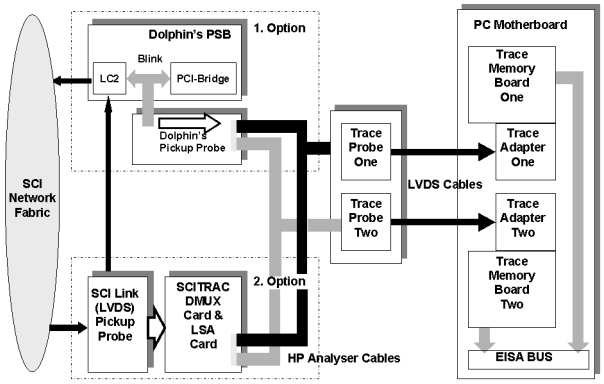

Fig. 2. Trace hardware overview

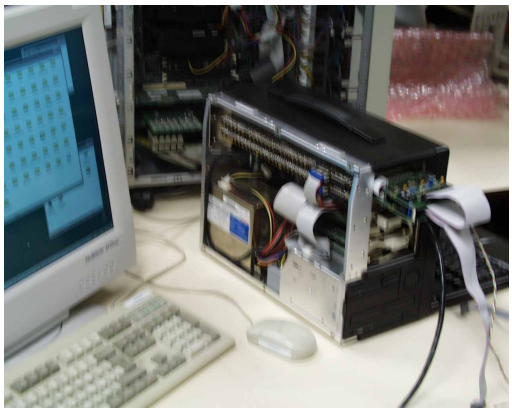

Fig. 3. Trace Instrument

I. Trace Instrument Hardware

The hardware of the trace instrument [1] comprises of a portable PC, two deep trace memory boards, two probe adapters [2] and two trace probes (see Fig. 2. and Fig.3).

Blink traces from Dolphin's SCI-PCI bridge can be acquired via a probe card supplied by Dolphin that attaches to their SCI interface cards via elastomeric connectors. This card breaks out the Blink signals to a number of connectors that will accept cables for a HP16500 series logic analyser (see Fig. 2, Option 1). Furthermore SCILAB's SCITRAC cable tracer provides broadly similar connectivity (see Fig. 2, Option 2).

The instrument requires two trace probes [2] that attach to the trace target via HP16500 series compatible cables and are synchronised by an inter-probe cable. Each trace probe attaches to 48bits of the 96bit-sample data path. The trace probe multiplexes the trace samples onto LVDS cables, which connect the probes to the trace instrument's adapter cards.

The probe adapters demultiplex the trace probes' LVDS signals. Each adapter attaches to one of the deep trace memory boards and provides the memory board with 48bits of a 96bit data path.

The deep trace memory boards are inserted into the PC's I/O slots. Each trace board contains 12 Mbytes of dual ported VRAM; one port receives trace data from the probe adapter while the second connects the trace memory to the I/O bus of the trace instrument.

The first trace board inserts absolute time stamps following each packet into the trace memory while the second board inserts relative time stamps.

II. Trace Instrument Software

The trace board's operation is controlled by a suite of driver, API and GUI application software through the I/O bus (see Fig. 5.). The trace tool API may be employed by the user to adapt the instrument to their specific needs.

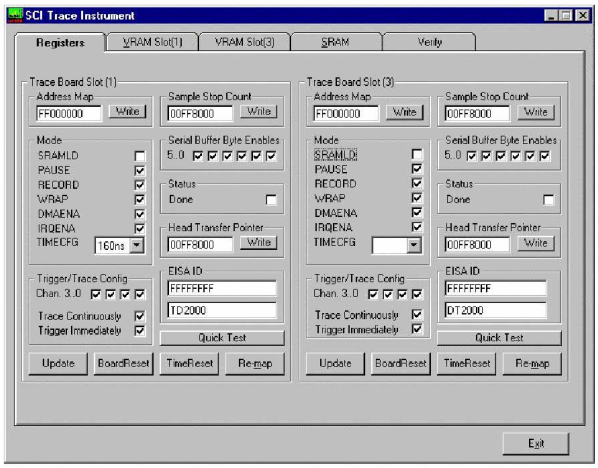

Fig. 4. Trace instrument control GUI

Both boards are interconnected to enable triggering over the full 96-bit sample width. Furthermore, it is intended that a number of instruments could be interconnected for a synchronised trace data acquisition on two or more target nodes. The trigger mechanism provides four level triggering. The filter and trigger patterns are configured through the instrument's API. A trigger and filter GUI implementation is shown in Fig.5.

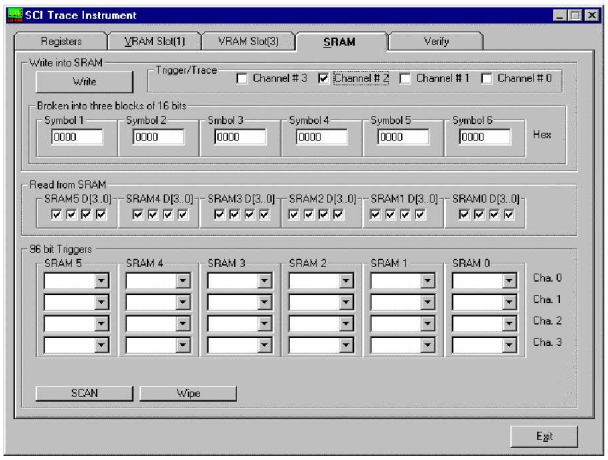

Fig. 5. Trace instrument trigger and filter GUI

A view of the trace board memory contents is provided through the instrument's control software (see Fig. 6.).

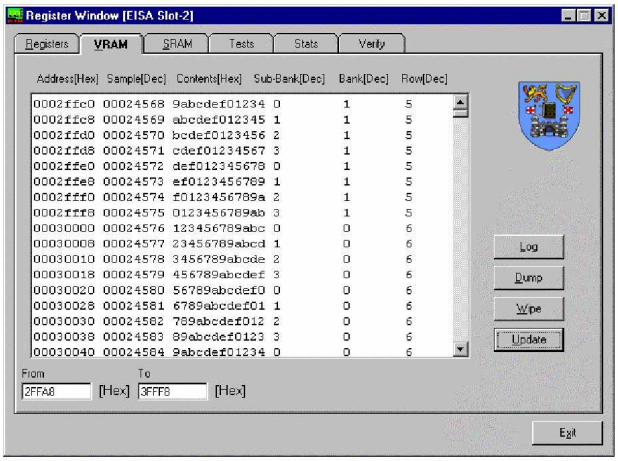

Fig. 6. Trace memory viewer

III. TRACE DATABASE

The trace instrument employs a relational database to store and analyse trace data [2]. The trace database is designed to accommodate all SCI packet types encountered on SCI cable links and Blinks. The following packet classification satisfies the Blink specification [3] and the SCI IEEE standard [4]. This categorisation is used for the decoding, the trace database storage and the retrieval of SCI and Blink packets.

A. *SCI cable links*

Type 1	Request-send-packet with extended header and 0 byte data
Type 2	Request-send-packet with extended header and 16 byte data
Type 3	Request-send-packet with extended header and 64 byte data
	•
	•
	•
Type 17	Response-send-packet with 256 byte data
Type 18	Response-echo-packet
Type 19	Idle Symbols
Type 20	Sync packets

B. *Blinks*

Type 21	Encapsulated request-send-packet with extended header and 0 byte data
Type 22	Encapsulated request-send-packet with extended header and 16 byte data
Type 23	Encapsulated request-send-packet with extended header and 64 byte data
	•
	•
	•
Type 34	Encapsulated response-send-packet with 16 byte data
Type 35	Encapsulated response-send-packet with 64 byte data
Type 36	Encapsulated response-send-packet with 56 byte data

Subsequent to a trace acquisition the instrument's control software writes the trace memory contents into two trace files. A Java decoding application reads the trace-data from those trace files and reunites the two 48bit fractions into a full 96bit-sample.

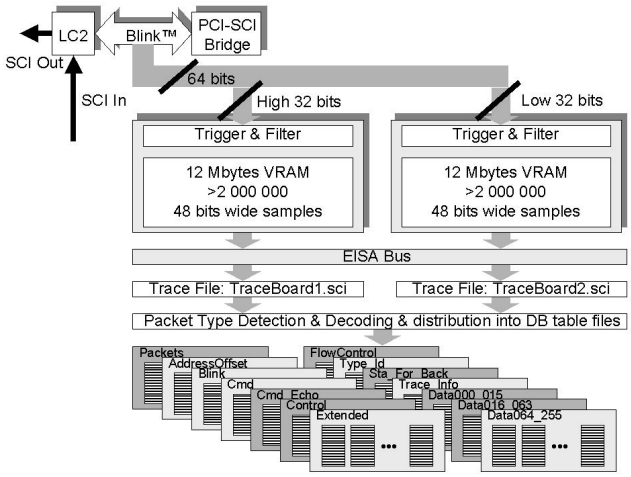

Fig. 7. Trace data flow from Blink into DB-table-files

The software also detects the packet types as categorised above and decodes the packets. The trace database is broken up into a number of tables to accommodate the various types of SCI packets. The database design provides space-optimised storage. The decoded SCI-packets are written into trace-database-table-files according to their packet type specification. These trace-database-table-files are used for a subsequent bulk import into the trace database. Each trace-database-table-file is associated with a table in the trace database. The file format reflects the database table design to accommodate bulk imports. Figure 7 demonstrates how trace data flows from a target node's Blink into the trace-database-table-files. Figure 8 gives an example of how a specific packet type, in this case a

Response-send-packet with 64 bytes data - Type 16, is distributed into the appropriate trace-database-table-files.

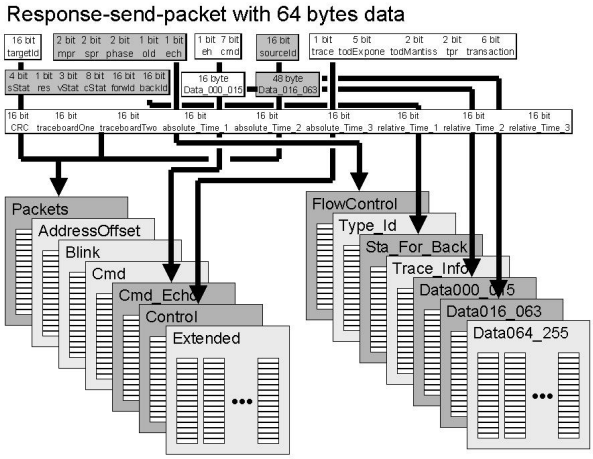

Fig. 8. Packet trace database distribution

A Trace-ID and a Packet-ID uniquely identify every SCI packet in every trace. Every trace-data-table contains these two IDs as primary keys. A main table is shared by all packets and contains a packet-type-ID but all packets occupy only a subset of the available tables. Figure 9 shows the relations between the trace database tables.

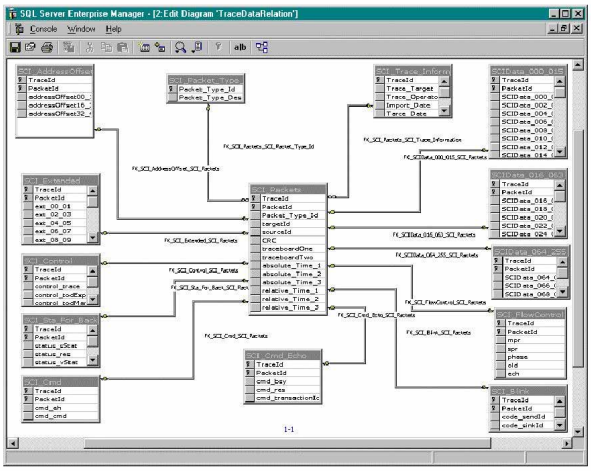

Fig. 9. Trace database relations

The fields in the trace-database table exhaustively enumerate SCI-packet information, preserving the maximum level of detail, e.g. targetID, command type, sourceID, etc. This allows for very detailed queries, e.g. all request-send packets with targetId = X, sourceId = Y and addressOffset between A and B. It is expected that an extensive query set will be accumulated as time goes by, some within specialised GUIs. The user may give meaningful interpretation to trace data fields through the implementation of additional tables and additional one-to-many relations.

The design allows the analysis of subsets of the packet's data while maintaining a relation to the full packet information e.g. a query result-set is easily associated with the full packet information.

C. *Trace database Performance*

A preliminary investigation has shown that direct SQL insertions of individual packets subsequent to the packet's decoding are too expensive. The estimated execution time exceeds 1 hour for a full trace while a bulk import into the trace instrument's MS Access database can be achieved in less than 10 minutes. MS SQL-Server imports are expected to be even less time consuming.

The following SQL-query reconstructs a specific *Type 1 Request-send-packet with extended header and 0 bytes data.* The packet has a TraceID = 3 and a PacketID = 40200 and is retrieved from a trace database with 100,000 packets. The query must retrieve 39 fields in 6 tables in order to reassemble this packet and it's associated trace information.

```
CREATE PROCEDURE [SCI_Packet_Type_01] AS SELECT
    SCI_Packets.TraceId,
    SCI_Packets.PacketId,
    SCI_Packets.Packet_Type_Id,
    SCI_Packet_Type_Id.Packet_Type_Description,
    ... (place holder 33 fields in 6 tables)
    SCI_Packets.relative_Time_2,
    SCI_Packets.relative_Time_3
FROM SCI_Packets
  INNER JOIN SCI_FlowControl ON
    SCI_Packets.TraceId = SCI_FlowControl.TraceId AND
    SCI_Packets.PacketId = SCI_FlowControl.PacketId
  INNER JOIN SCI_Cmd ON
    SCI_Packets.TraceId = SCI_Cmd.TraceId AND
    SCI_Packets.PacketId = SCI_Cmd.PacketId
  INNER JOIN SCI_Control ON
    SCI_Packets.TraceId = SCI_Control.TraceId AND
    SCI_Packets.PacketId = SCI_Control.PacketId
  INNER JOIN SCI_AddressOffset ON
    SCI_Packets.TraceId = SCI_AddressOffset.TraceId AND
    SCI_Packets.PacketId = SCI_AddressOffset.PacketId
  INNER JOIN SCI_Extended ON
    SCI_Packets.TraceId = SCI_Extended.TraceId AND
    SCI_Packets.PacketId = SCI_Extended.PacketId
  INNER JOIN SCI_Trace_Information ON
    SCI_Packets.TraceId = SCI_Trace_Information.TraceId
  INNER JOIN SCI_Packet_Type_Id ON
    SCI_Packets.Packet_Type_Id = SCI_Packet_Type_Id.Packet_Type_Id
WHERE (SCI_Packets.TraceId = 3) AND
      (SCI_Packets.PacketId = 40200)
```

The query was executed using a Microsoft SQL-Server 7.0 on a 450 MHz Intel Pentium II with 128 MB memory and required less than 1second. The same query into an MS Access database requires about 15 seconds. These preliminary results indicate that a Microsoft SQL-Server is a suitable database engine for a Packet Viewer (see Fig. 11 for an example of a packet viewer applet).

IV. Trace Data Presentation and Analysis

The primary trace data acquisition, the decoding and the trace bulk import are associated with the trace instrument itself. But trace data will in all likelihood be transferred to a remote note for performance and accessibility reasons. Figure 10 provides a system overview. Trace data is easily transferred from one trace database to another. A remote node hosts a web server and a Java trace database server. Client nodes may load trace viewer and analysis applets into their web browser. The trace instrument can behave as a client in this scenario. The applet establishes a socket connection to the trace database server.

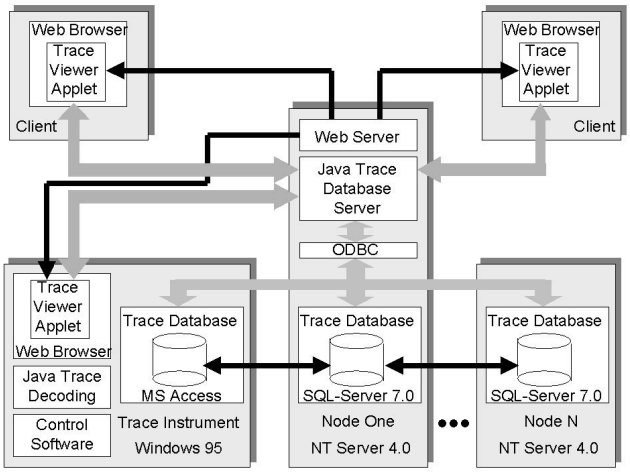

Fig. 10. Trace system software

A. *Java Trace Database Server*

The Java database server creates a new thread for every connecting client, thereby allowing concurrent access from multiple clients. The client applet initiates the server to connect to a particular trace database either on the local node or a remote node. The Java server establishes the trace database connection through an ODBC server. The trace database server holds a set of prepared SQL statements. A client may invoke a specific prepared SQL statements and forward parameters to the server. The server then invokes the statement with the inserted client parameters and returns the query result-set to the applet.

B. *Java Packet Viewer Applet*

Figure 11 shows a SCI packet viewer applet. The user provides the applet with a TraceID and PacketID. The software then initially queries the packet type and adjusts its layout accordingly. A subsequent type-specific query for the full set of trace-data provides the applet with the required data.

V. Conclusion and Further Work

This trace instrument provides a non-intrusive method of measuring SCI interconnect traffic and consequently will not influence the temporal behaviour of the system. It will enable researchers and developers to analyse the true temporal behaviour of clusters made up of standard components. The employment of a relational database for trace-data storage provides the user with well understood and easy-to-use tools to extend and to adapt the predefined database queries to their specific needs. The use of Java and SQL makes the software platform independent.

The prototype will be enhanced through the implementation of important methods for the analysis and visualisation of the dynamic behaviour of parallel processes [10]. It is proposed to employ time state diagrams (gantt charts) and causality diagrams (hasse diagrams). SCILAB Technology AS will incorporate the software into its SCIview instrument [7] under a nonexclusive license agreement.

The instrument is a vital tool for the validation of global state estimation algorithms. In this context Trinity's research interest is aimed at the runtime optimisation of DSM systems.

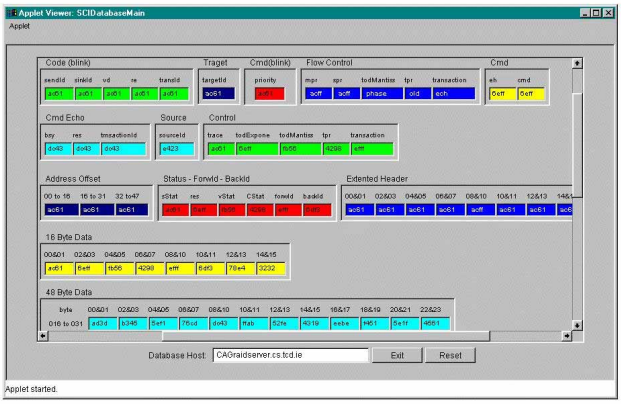

Fig. 11. Java Packet Viewer

References

[1] Coghlan, B.A., Manzke, M., Barnstedt, E., Cunniffe, R., Dukes, J., **Deep Trace DT200.1**, Prototype Tracer, technical manual, http://www.cs.tcd.ie/Brian.Coghlan/scieuro/scieuro.htm, Esprit Project 25257, SCI Europe, Work package, Trinity College Dublin, 1998

[2] Coghlan, B.A., Manzke, M., **Prototype Trace Probe and Probe Adapter and Prototype Tracer Software**, http://www.cs.tcd.ie/Brian.Coghlan/scieuro/scieuro.htm, Esprit Project 25257, SCI Europe, Deliverables, Trinity College Dublin, 1999

[3] Dolphin: **A Backside Link (Blink™) for Scalable Coherent Interface (SCI) nodes**, Draft 2.41, 1996

[4] IEEE 1596, **IEEE Standard for Scalable Coherent Interface (SCI)**, IEEE Std 1596-1992, IEEE Computer Society, Aug.1993.

[5] Manzke, M. Coghlan, B., **Non-Intrusive Deep Tracing of SCI Interconnect Traffic**. in Proceedings of SCI-Europe, September 2nd and 3rd, 1999 held as a Conference Stream of Euro-PAR'99 Toulouse, France, 31 August to 3 September 1999

[6] Karl, W., Leberecht, M., **Ein Monitorkonzept für Systeme mit verteiltem gemeinsamen Speichern**, ARCS'97: Architektur von Rechensystemen, Sept. 1997.

[7] Skaali, B., Baard, N., Birkeli, I., Wormald, D., **SCIview-SCI Test, Verification and Monitoring Instrument**, in Proceedings of SCI-Europe, September 2nd and 3rd, 1999 held as a Conference Stream of Euro-PAR'99 Toulouse, France, 31 August to 3 September 1999

[8] SCIEurope, Esprit Project 25257, http://www.oslo.sintef.no/ecy/projects/SCI_Europe/index_net.html

[9] Skaali, B., Birkeli, I., Nossum, B.A., Wormald, D., **SCITRAC – an LSA Preprocessor for SCI Link Tracing,** in Proc. SCI Europe, Bordeaux, France, Sept. 1998.

[10] Klar, R., Dauphin, P., Hartleb, F., Hofmann, R., Mohr, B., Quick, A., Siegle, M., **Messung und Modellierung parralleler und verteilter Rechensysteme,** ISBN 3-519-02144-7, Teubner Stuttgard, 1995.