Implementation of Estelle Specifications on the KSR1–32

Stefan Fischer

University of Mannheim, Praktische Informatik IV, P.O. Box 10 34 62,
D-68131 Mannheim, Germany

fischer@pi4.informatik.uni-mannheim.de

1. Introduction

Existing communication protocol suites such as the ISO/OSI [7] or the INTERNET protocols [2] were designed with relatively slow networks in mind. The end systems were fast enough to process complex protocols because the data transmission time was long compared to the time needed for protocol execution.

Given the fast transmission media based on fiber optics that are now available, current end systems are too slow for high performance communication. Communication software has become the major bottleneck in high speed networks [10, 1]. Efficient implementation of the protocol stack is of crucial importance for the networking future.

For specification purposes, formal description techniques are now widely used. They improve the correctness of specifications by avoiding ambiguities and by enabling formal verification. In addition, they allow semiautomatic code generation.

This technique has several advantages: The code can be maintained more easily since the system is specified in an abstract, problem–oriented language. It is also much easier to port an implementation to another system. But one of the major problems is the performance of implementations produced automatically from a formal specification.

In our project we use the formal description technique Estelle [8]. Estelle specifications mainly consist of a hierarchically ordered set of extended finite state machines, so–called modules, which are communicating by message exchange via directional channels with queues on each end.

Existing code generators were made to easily get rapid prototypes for simulation purposes. They produce higher language code (e.g. C or C++) from Estelle specifications. Executable specifications lead to a better understanding of protocol behaviour. Since performance aspects are not essential for such a simulation, existing Estelle tools are designed for validation rather than for the generation of efficient code for high performance implementations.

A considerable amount of the runtime of automatically generated implementations is spent in the runtime system. This gives rise to hopes that much more efficient implementations can be created by code generators if the runtime system can be improved, especially by the use of parallelism.

Thus, the goal of our project was to develop a code generator that accepts an Estelle specification as input and produces a parallel program as output.

To use a parallel protocol implementation efficiently, it is important to have a multiprocessor system. For our project, we selected a KSR1 [5] equipped with 32 processors and running the operating system OSF/1. This machine is available at the University of Mannheim since December 1992. The most important characteristics for our project were, in case of the machine, the virtual shared memory architecture, which reduced the complexity of the code generator, and, in case of the operating system, the availability of threads which leads to a better performance by reducing the context switching overhead.

2. Structure of Implementations

In this project we did not implement a fully new Estelle compiler. We used the PetDingo system [9] and exchanged the runtime system. A detailed description of the new system may be found in [4] and [3].

The structure of the generated code (C++) remains the same. In the old runtime system there existed a scheduler which called one Estelle module after the other while observing a certain Estelle semantics. In the new system, each module is run by one thread thus allowing the use of multiple processors. The above mentioned Estelle semantics is observed by the use of explicit module synchronization. There is no central scheduler anymore.

For the synchronization we use OSF/1 lock and condition variables. When a module wants to synchronize with another one, it changes the state of a shared memory variable and signals this change using a condition variable. The partner module will be waked up and read the new state of the shared memory variable. It will then execute its own cycle until it is ready. Parallelism is possible, as one module may synchronize with more than one other module at a time, if Estelle semantics allows it.

3. Performance Results

For measuring the speedup of parallel implementations of a protocol in comparison to a sequential one, we set up the following scenario: the basic building block is a connection between an initiator and a responder. They are each sitting above a quite generalized protocol stack of a dynanmic height that might be adjusted from 1 to 5. Each module in this stack takes a message from its input queue, does some processing and puts the message to his output queue. Both stacks are connected by a transport pipe that delivers messages from one stack to the other. A supervisor module may create 1 to 5 of these connections. Thus, we are able to measure both the influence of processor–per–layer and processor–per–connection parallelism (for a deeper description of these terms, see [6]).

For our measurements we used 20 of the 32 processors of the KSR1. The results may be seen in Fig.1. For this protocol, we obtain a remarkable speedup of about 10 to 11 with 5 connections and a protocol stack height of 5. It has to be noted that the full speedup (20 in this case because of 20 processors) will never be reachable in the implementation of Estelle specifications. This is due to Estelle semantics which explicitly forbids full parallelism between all modules. The most important barrier for a further speedup is the rule that modules on different levels in the same tree of the module hierarchy (for a more precise description of this concept see [8]) may never run in parallel. The synchronization rules of Estelle specifications are quite complex; ongoing research in our group deals with a further increase of the speedup.

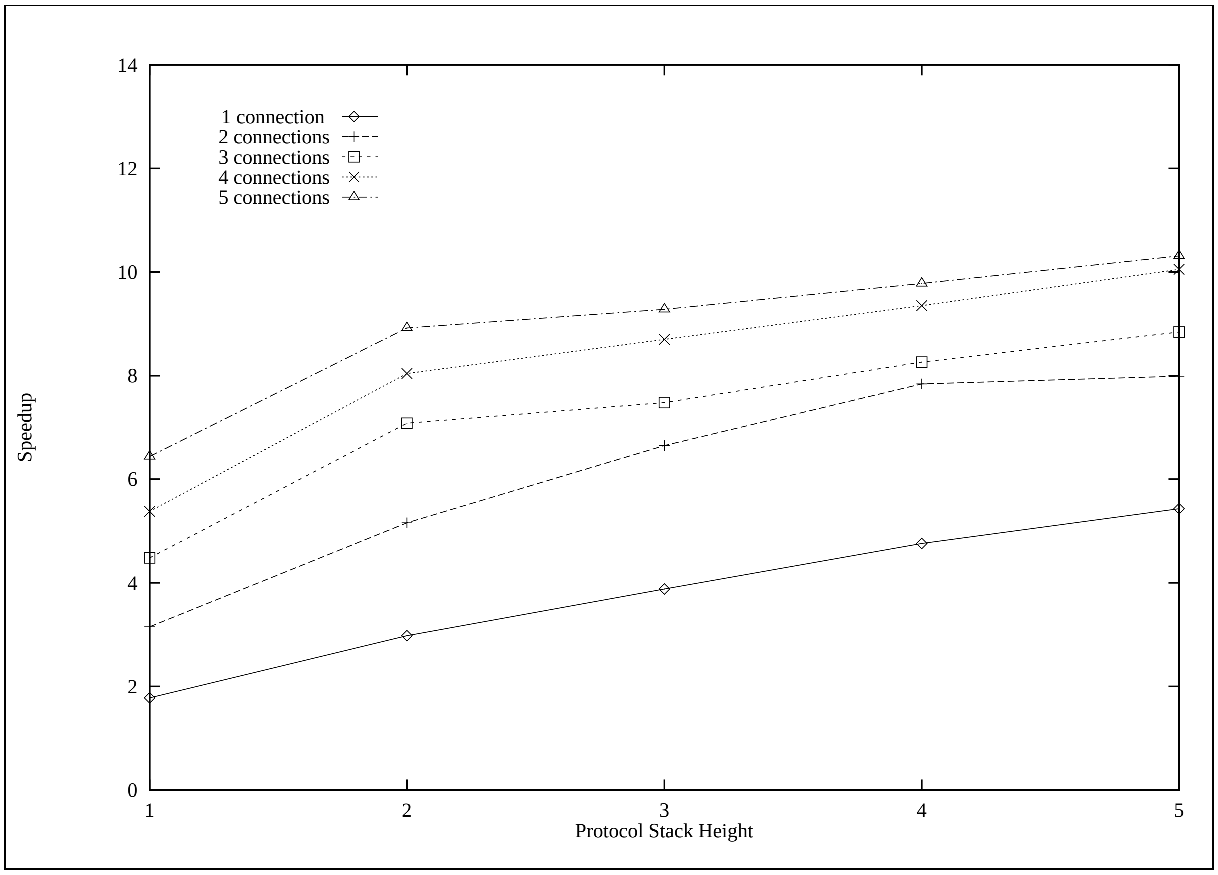

Figure 1. Performance Results for Multiple Connections

It is obvious that the increase of the number of connections leads to an increase of performance. An interesting point is the structure of the single curves, especially when using 3, 4 or 5 connections. The curve rises steeply when using only a small stack height and flattens when the height is increased. The reason for this is that, at this point, the number of threads becomes greater than the number of available processors. The processor utilization becomes better, but also the the sum of synchronization times between threads increases.

4. Future work on the KSR1

In the project described above we are currently working on an improved version of the Estelle compiler. In another project, we have ported an MPEG encoder to the KSR. MPEG is a standardized compression format for digital movie streams. We are investigating optimal parallelization units for the encoding of MPEG streams.

REFERENCES

1. D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *SIGCOMM '90 Symposium Communication Architectures & Protocols*, pages 200–208, Philadelphia, September 1990.
2. D. Comer. *Internetworking with TCP/IP*. Prentice–Hall, Englewood Cliffs, 1988.
3. S. Fischer. Generierung paralleler Systeme aus Estelle–Spezifikationen. Master's thesis, Lehrstuhl für Praktische Informatik IV, Universität Mannheim, 1992.
4. Stefan Fischer and Bernd Hofmann. An Estelle Compiler for Multiprocessor Platforms. In Richard L. Tenney, Paul D. Amer, and Ümit Uyar, editors, *Formal Description Techniques VI, Boston, USA*. Elsevier Science Publishers B.V. (North-Holland), October 1993. to appear.
5. S. Frank, H. Burkhard III, and J. Rothnie. The ksr1: High performance and ease of programming, no longer an oxymoron. In H.-W. Meuer, editor, *Supercomputer '93: Anwendungen, Architekturen, Trends*, Informatik aktuell, pages 53–70. Springer Verlag, Heidelberg, 1993.
6. B. Hofmann, W. Effelsberg, T. Held, and H. König. On the Parallel Implementation of OSI Protocols. In *IEEE Workshop on the Architecture and Implementation of High Performance Communication Subsystems*, Tucson, Arizona, February 1992.
7. Information processing systems — Open Systems Interconnection — Basic Reference Model. International Standard ISO 7498, 1984.
8. Information processing systems — Open Systems Interconnection — Estelle: A formal description technique based on an extended state transition model. International Standard ISO 9074, 1989.
9. Rachid Sijelmassi and Brett Strausser. The PET and DINGO tools for deriving distributed implementations from Estelle. *Computer Networks and ISDN Systems*, 25(7):841–851, 1993.
10. L. Svobodova. Measured performance of transport service in LANs. *Computer Networks and ISDN Systems*, 18(1):31–45, 1989.