Turbo Erlang: An Efficient Implementation of A Concurrent Programming Language

(Extended Abstract)

Bogumil Hausman

Computer Science Laboratory
Ellemtel Telecommunications Systems Laboratories[1]
Box 1505, S-125 25 Älvsjö, Sweden
Phone: +46 8 727 32 67
Fax: +46 8 647 82 76
{bogdan}@erix.ericsson.se

Introduction

Erlang is a concurrent programming language designed for prototyping and implementing reliable real-time systems. Erlang was developed at the Computer Science Laboratory, Ellemtel Telecommunications Systems Laboratories [3, 4]. Erlang provides support for programming concurrent applications, a special syntax for referring to time (time-outs), and explicit error detection capabilities. Erlang in its design inherits some ideas from concurrent logic programming languages. It does not allow destructive assignment of variables and uses pattern matching for variable binding and function selection.

In this paper we describe a very efficient and portable sequential implementation of Erlang where Erlang programs are compiled into the C language. We call the implementation Turbo Erlang.

The proposed implementation technique can be easily applied to other high-level language implementations (e.g. Janus [6] and KL1 [5]).

The Turbo Erlang Abstract Machine (TEAM)

The construction of the turbo Erlang abstract machine (TEAM) has been influenced by ideas behind the conventional WAM [8, 9] and the Janus Virtual Machine (Janus compiled to C) [6]. Differences come from the fact that Erlang has a functional nature, has fully ground terms, and has explicit notions of concurrency and time.

Comparing further to the Janus Virtual Machine, we use GCC instead of cc which allows global register allocation and simplifies return address administration as in GCC labels can be used as first-class objects. There are some differences in data object representation and our system provides a copying garbage collector while the Janus Virtual Machine has no garbage collection implemented yet. Another difference is that in our system each Erlang module is compiled into a C function residing in a separate file, and modules can be dynamically loaded at run-time. In the Janus system all programs are compiled into one C procedure which does not allow for large-scale applications.

Evaluation Results

To have the possibility to compare with evaluation results for the Janus Virtual Machine our benchmark programs are those used in [6].

In Table 1, the time reported (in milliseconds) is the time to execute the program once.

[1] a company jointly owned by Ericsson and the Swedish PTT.

Program	TEAM(T)(ms)	Janus(J)(ms)	J/T	SICStus (S)(ms)	S/T
hanoi	104	182	1.75	291	2.79
tak	179	207	1.49	759	4.24
nrev	1.19	0.729	0.61	1.78	1.49
qsort	1.81	2.03	1.12	3.55	1.96
factorial	0.0418	0.0393	0.94	0.198	4.73

Table 1: The performance of Turbo Erlang (the garbage collection time not included) compared to Janus and SICStus Prolog.

Conclusions

We have presented an efficient and portable sequential implementation of Erlang (called Turbo Erlang), where Erlang programs are compiled into the C language. Portable means that the implementation can be ported to any processor having a GCC compiler. We have chosen GCC to allow global register allocation and usage of labels as first-class objects, which in turn simplified return address administration, and generation of unique labels.

The Turbo Erlang system performs very well in comparison with other high-level language implementations like SICStus Prolog (compiling to native code) and jc (Janus compiled to C).

Comparing further to Janus, in our system each Erlang module is compiled into a C function residing in a separate file and the module functions can be accessed from other modules with no overhead. In the Janus system all programs are compiled into one C procedure which does not allow for large-scale applications.

The comparison with C shows that the performance of Turbo Erlang is not very far from the performance of highly optimized C code. We expect that after applying some compile-time optimizations we will be able to run large applications in Erlang as quickly as the same applications written from the beginning in C or C++.

Acknowledgements

We would like to thank Robert Virding and Mike Williams for discussions considering implementation aspects of Erlang. The C level dynamic link editor was written at our laboratory by Torbjörn Törnkvist. We are also grateful to David Gudeman for providing his benchmark programs.

References

[1] Ahlberg, I., Danne, A., and Bauner, J-O. Prototyping Cordless Using Declarative Programming. *XIV International Switching Symposium*. Yokohama, 1992.

[2] Armstrong, J.L., Däcker, B.O., Virding, S.R., and Williams, M.C. Implementing a Functional Language for Highly Parallel Real Time Applications. *Proceedings of SETSS'92*. Florence, 1992.

[3] Armstrong, J.L., and Virding, S.R. Programming Telephony. In *Strand - New Concepts in Parallel Programming*, pages 289–304. Foster, I., and Taylor, S. Prentice Hall, 1990.

[4] Armstrong, J.L., Virding, S.R., and Williams, M.C. Concurrent Programming in Erlang. To be published. Prentice Hall, 1993.

[5] Chikayama, T., Tetsuro, F., and Yashiro, H. A Portable and Reasonably Efficient Implementation of KL1. Draft version. ICOT Research Center, Japan, 1993.

[6] Gudeman, D., De Bosschere, K., and Debray, S. jc: An Efficient and Portable Sequential Implementation of Janus. *Joint International Conference and Symposium on Logic Programming*, pages 399–413. MIT Press, 1992.

[7] SICStus Prolog User's Manual, Version 2.1 # 6. Swedish Institute of Computer Science, August, 1992.

[8] Warren, D.H.D. An Abstract PROLOG Instruction Set. SRI Technical Note 309. October, 1983.

[9] Warren, D.H.D. PROLOG Implementation and Architecture. *3rd International Logic Programming Conference*, Tutorial Notes. London, 1986.