

Los Alamos National Laboratory is operated by the University of California for the United States Department of Energy under contract W-7405-ENG-36

LA-UR--84-4049

DE85 0056 45

TITLE PORTABLE STANDARD LISP ON THE CRAY

AUTHOR(S) James W. Anderson, C-10

SUBMITTED TO To be published in a special issue of SLAM during mid-1985. The papers to appear in this issue were presented at the ARO workshop on New Computing Environments: Parallel, Vector and Systolic held at Stanford November 7 - 9, 1984.

DISCLAIMER

This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor any agency thereof, nor any of their employees, makes any warranty, express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof.

By acceptance of this article, the publisher recognizes that the U.S. Government retains a nonexclusive, royalty free license to publish or reproduce the published form of this contribution, or to allow others to do so, for U.S. Government purposes.

The Los Alamos National Laboratory requests that the publisher identify this article as work performed under the auspices of the U.S. Department of Energy.

MASTER

Los Alamos Los Alamos National Laboratory Los Alamos, New Mexico 87545

Portable Standard LISP on the Cray

J. Wayne Anderson*

Abstract. Portable Standard LISP, a dialect of LISP developed at the University of Utah, has been implemented on the Cray-1s and Cray X-MPs at the Los Alamos National Laboratory and at the Magnetic Fusion Energy Computer Center at Lawrence Livermore National Laboratory. This was done in order to provide an environment for symbolic processing on the Crays and to evaluate how well the Crays support this environment. This implementation is discussed and some performance results presented.

1. **Introduction.** Research toward developing a portable LISP system received impetus in 1966 [4] when a model for a standard LISP subset was developed to make the REDUCE [5] symbolic arithmetic package more portable. The research effort at the University of Utah has since produced progressively larger and more portable subsets of LISP [2], the most recent of which is Portable Standard LISP (PSL).

Among the goals of the designers of PSL were: provide a uniform LISP programming environment across a spectrum of machines; produce a portable system comparable in execution speed to other non-portable LISP systems; and effectively support REDUCE on different machines.

PSL is currently being distributed for DECSys-20s, VAXs running under both UNIX and VMS, HP9836s, and APOLLOs. PSL is ready for distribution for Crays running the CTSS operating system and is being developed for Crays running COS. It is also being developed for the IBM 3081.

Tables 1 and 2 illustrate the execution speed of PSL relative to that of other versions of LISP. Benchmark programs developed by Gabriel [1] were executed and run on a VAX 11/780 using various dialects of LISP. For the sake of brevity, tables 1 and 2 present results from just a few of the benchmarks. The results presented, however, are typical. An entry of "-" means that a benchmark was not able to execute in a dialect of LISP.

The following is a brief description of the benchmark programs:

BOYER - a theorem prover benchmark featuring

*Los Alamos National Laboratory, Los Alamos, NM 87545

LISP structure manipulations and having a typical number of function calls;

- BROWSE - an expert system benchmark with pattern matching and frames for knowledge application storage;
- DESTRUCT - a benchmark with destructive list operations such as REPLACA and REPLACD;
- STak - a program that times function calls with fluid bindings;
- PUZZLE - a game with vector references; and
- TRIANG - a boardgame benchmark.

There were several reasons for wanting to have LISP on

Table 1

	INTERLISP	COMMONLISP	FRANZLISP	PSL
BOYER	53.3	87.7	71.5	41.3
BROWSE	111.5	205.0	170.3	50.3
DESTRUCT	5.4	6.4	13.7	3.9
STak	9.7	4.1	6.3	5.4
PUZZLE	110.3	47.5	-	16.3
TRIANG	1076.5	360.9	-	212.2

Real Time in Milliseconds
VAX 11/780

Table 2

	INTERLISP	COMMONLISP	FRANZLISP	PSL
BOYER	1.3	2.1	1.7	1.0
BROWSE	2.2	4.0	3.4	1.0
DESTRUCT	1.4	1.6	3.5	1.0
STak	2.4	1.0	1.5	1.3
PUZZLE	6.8	2.9	-	1.0
TRIANG	5.1	1.7	-	1.0

Normalized Execution Times
(shortest execution time = 1.0)
VAX 11/780

the Crays at Los Alamos National Laboratory. One reason was the interest in having symbolic programming environments on the most powerful machines available. This could provide the capability to solve symbolic problems that would not be feasible to solve on less powerful systems. There also was, and there still is, interest in the possibility of combining symbolic programming with some of the large numeric codes.

Three primary considerations led to the selection of PSL from among the dialects of LISP. As one of the design criteria claimed for PSL was that of portability, it appeared that it could be implemented more quickly than other dialects. The selection of PSL also provided the symbolic arithmetic capabilities of REDUCE. Finally, PSL appeared to be an efficient dialect of LISP.

2. Implementation. In June 1982 a meeting was held at the University of Utah to outline the Cray implementation effort. By July of 1984 PSL interpreters and compilers were available for use on all Cray-1s and Cray X-MPs at the Los Alamos National Laboratory. Soon thereafter, PSL was also available on the Crays at the National Magnetic Fusion Energy Computer Center (MFECC) of Lawrence Livermore National Laboratory. REDUCE was subsequently implemented at both sites.

The effort required to implement PSL on the Crays, while non-trivial, was much less than that required to implement a non-portable dialect. In order to understand the implementation procedure, it is first necessary to take a look at the steps involved in PSL compilation [3].

LISP code is first translated into instructions for an Abstract LISP Machine (ALM). The ALM is a general-purpose register machine whose instructions are expressed in LISP Assembly Program (LAP) format. The LAP format has the form:

ALMopcode ALMoperand...ALMoperand

ALMopcodes are referred to as cmacros and an instruction may have zero or more operands. The ALM in this instance has 15 general purpose registers. The next step is to translate the ALM instructions into Target LISP Machine (TLM) instructions. The TLM instructions are also in LAP format with opcodes and operands of the TLM.

From here there are three separate paths that can be taken in the compilation process. The TLM instructions can be translated:

1. into machine code and placed into the memory

of a running PSL system;

2. into machine code and saved in a file for loading at a later time into a running PSL system; or

3. into assembly language for later assembly on a target machine.

This third path is the one taken when bootstrapping PSL from a host system to a target machine. At Los Alamos the host system used for this purpose was a VAX 11/780 running UNIX.

In the bootstrapping process, it is necessary to modify the PSL compiler on the host machine for it to become a cross compiler for the target machine. Tables must be supplied that will be used in the translation of ALM opcodes and operands into TLM opcodes and operands. In order to simplify new machine implementation and increase the amount of common code, all translations from one code form to another have been made extremely table driven. The necessary formats, constants, and procedures must be supplied for translating the TLM instructions into the host machine's assembly language. The closer the TLM instruction format is to that of the host machine's assembly language the easier the translation will be. Unfortunately, in the case of the Cray, the match was not close. For example, consider the following Cray assembly language (CAL) instruction:

S1 S2 + S3

This adds the contents of register S2 to that of register S3 and stores the result in register S1. Thus the format of CAL instructions is along the lines of:

destination operand opcode operand

This resulted in one extra step in the translation process. The TLM instructions were mapped into an intermediate form that more closely matched the CAL format before further translation.

Some support code had to be written on the Cray to interface the cross compiled code to Cray system functions, for example, input/output. After the cross compiled code is assembled on the Cray it is linked with the support code for execution.

There is a carefully graded set of tests that are used in the bootstrapping process. Each test results in an ever increasing subset of PSL being cross compiled,

shipped to the Cray, assembled, and executed. A test that fails results in modifications to the cross compiler or other processes used in the implementation procedure. That portion of PSL that is successfully tested is then used as a basis for succeeding tests. After all tests are executed, the major portion of PSL has been implemented.

3. Performance. Once PSL had been successfully implemented on the Cray, Gabriel's benchmarks were executed and the results were compared to their execution on other machines. Tables 3 and 4 summarize the results of a typical subset of benchmarks. The benchmarks were also coded in ZETALISP and executed on a SYMBOLICS 3600. Table 5 compares the execution on the 3600 to that on the Cray.

Table 3

	Cray	VAX 11/780	DEC 20	S1	IBM 3081
BOYER	3.4	41.3	23.6	10.0	4.6
BROWSE	8.4	50.3	28.7	10.2	6.3
DESTRUCT	0.4	3.9	2.4	0.9	-
STak	1.1	5.4	2.7	4.3	1.7
PUZZLE	1.0	16.3	15.9	1.8	1.5
TRIANG	14.4	212.2	86.9	62.1	25.4

Real Time in Milliseconds
PSL

Table 4

	Cray	VAX 11/780	DEC 20	S1	IBM 3081
BOYER	1.0	12.3	7.0	3.0	1.4
BROWSE	1.3	8.0	4.5	1.6	1.0
DESTRUCT	1.0	8.8	5.4	2.1	-
STak	1.0	4.8	2.4	3.8	1.5
PUZZLE	1.0	16.3	15.9	1.8	1.5
TRIANG	1.0	14.5	6.0	4.3	1.8

Normalized Execution Times
(shortest execution time = 1.0)
PSL

When implementing REDUCE on PSL, a set of test routines is executed. Table 6 presents the time required for the

Table 5

	Cray	3600
BOYER	1.0	5.5
BROWSE	1.0	5.0
DESTRUCT	1.0	8.4
STak	1.0	2.3
PUZZLE	1.0	13.9
TRIANG	1.0	10.5

Normalized Execution Times
(shortest execution time = 1.0)

execution of the test routines on several different systems.

3. Summary and Future Work. The PSL implementation on the Cray has been successfully completed. The performance studies indicate that it is one of the most powerful LISP environments currently available. However, all the power of the Cray has not been realized. In mapping from an ALM with 15 general purpose registers, it was extremely difficult to make efficient use of the many special purpose registers and vector processing capabilities of the Cray. This resulted in an implementation with many possible areas of optimization. Some areas under consideration now include: scheduling of registers; using temporary registers to hold the top items of a stack; and using the vector registers during garbage collection.

4. Acknowledgements. I would like to acknowledge the contributions made to the implementation effort by Bruce Curtiss of MFECC, Dana Dawson of CRI, and, especially, Robert Kessler of the University of Utah. Without the

Table 6

Cray	4 seconds
DEC 20	25 seconds
HP9836U	55 seconds
VAX 11/780	60 seconds
APOLLO	80 seconds
VAX 11/750	90 seconds

REDUCE TIMINGS

efforts and knowledge of Dr. Kessler, the PSL implementation on the Cray would not yet be completed. I would also like to thank Richard Gabriel for the many benchmarks and results he supplied and for allowing me to use them in this paper.

REFERENCES

- [1] R. P. GABRIEL, Evaluation and Performance of LISP Systems, (to be published).
- [2] M. L. GRISS, E. BENSON and G. Q. MAGUIRE, JR., PSL. a portable LISP system, The Proceedings of the 1982 ACM Symposium on LISP and Functional Programming, Carnegie-Mellon University, Pittsburgh, August 1982, pp. 88-96.
- [3] M. L. GRISS, E. BENSON, R. KESSLER, S. LOWDER, G. Q. MAGUIRE, JR., and J. W. PETERSON, PSL Implementation Guide, Utah Symbolic Computation Group, Computer Science Department, University of Utah, Salt Lake City, 1983.
- [4] A. C. HEARN, Standard LISP, SIGPLAN Notices 4,9 (September 1966).
- [5] A. C. HEARN, REDUCE 2 Users Manual, Utah Symbolic Computation Group Report UCP-19, Computer Science Department, University of Utah, Salt Lake City, 1973.