A Conservative Garbage Collector for an EuLisp to ASM/C Compiler

E.Ulrich Kriegel
Fraunhofer Institute for Software Engineering and Systems Engineering
Kurstrasse 33
D-10117 Berlin
Germany
ulrich.kriegel@isst.fhg.de

1 Introduction

In the framework of the joint research project Apply [Bretthauer *et al.*, 1992] at ISST we are investigating strategies for the compilation of EuLisp modules [Padget *et al.*, 1993] into Sparc assembler or into C code. The main goals of the project are:

- Compiled EuLisp modules should run as efficient as equivalent C programs.
- Lisp procedures should be easily callable from C and vice versa.

The following implementation decisions were taken:

1. The hardware stack is used instead of an own control stack for Lisp.
2. Lisp datatypes are represented similar to C datatypes in order to avoid incompatibility with hardware datatypes and in order to enable easy data exchange with non-Lisp programs.

To reduce the size of objects a BIBOP typing scheme is used for frequently allocated data and a special tagging scheme with a 4 byte tag in front of data otherwise. The memory management system for the EuLisp runtime system relies on the conservative pointer finding technique [Boehm and Weiser, 1988].
In the following a brief description of the memory management layout is given. First tests, performed with a preliminary version of an EuLisp to C code compiler show that conservative techniques could be used at least for small, stand alone Lisp applications.

2 Layout of the Memory Management Scheme

We decided to implement a scheme similar to that described by Boehm and Weiser where data are located on special memory locations - in the following we call them cards. In order to support a BIBOP typing scheme the original BW-algorithm was modified:

- Objects are grouped on cards according to their size *and* according to the class they are instances of.

- In order to reduce fragmentation, vector-like objects (e.g. strings) that belong to the same class and differ only in the number of elements can be grouped on special cards.
- In order to simplify marking algorithms either a simple geometric description of the stored objects (e.g. 8-byte-constituents, no pointer) or a marking procedure must be assigned to each card.

A prototypical implementation (written in C) was used in conjunction with procedures written in C[1] to carry out performance tests. Compared to the original algorithm memory allocation was less efficient. This is mainly due to a more complicated dispatch algorithm. The efficiency for marking cards was found to be comparable with Boehm and Weiser or even better in case that marking procedures had been assigned to cards.

Thus the memory management system was redesigned with the concept of a configurable toolkit in mind. It is implemented in C and the configuration can be specified in form of preprocessor macros.

- In order to reduce the overhead of type and object size dispatch we now use descriptors that are calculated only once for every object class. For the allocation of an instance of a specific class then the corresponding descriptor is used.
- For every class a tracing function must be specified, predefined trace functions are provided for standard cases.
- Locations can be declared to be root addresses. With declared roots the total root set consists of all stack locations and the declared root set. Without declaration all possible locations for static data are included into the root set.
- Different types of cards are supported in order to reduce possible fragmentation:

  STSS Single Type Single Size Cards
  : STSS cards allow the efficient storage of objects of same size and same type. The object class is stored among other relevant information in the card header. This scheme corresponds to the BEBOP typing scheme known in Lisp implementation and should be used when many instances of the same class will be created.

  MTSS Multiple Type Single Size Cards
  : MTSS cards contain objects of the same size that belong to different classes. The advantage over STSS-cards is a smaller fragmentation of memory specially in cases where only a small number of instances of each class will be created. The object class is stored in front of the corresponding data as in normal tagging schemes. This increases the object size by one word. However, in order to allow uniform access to the object slots independently of the used card type any valid pointer has to point behind the tag into the first slot of an object.

[1] We used a version of factorial using an arithmetic for natural numbers where each number N is represented as the Nth. successor of zero.

STMS Single Type Multiple Size Cards
STMS cards store variable sized vector-like objects like strings. Similar to the scheme used on MTSS cards the length of each instance is stored in front of that instance.

- For each application the user may specify which kind of cards he intends to use to allow optimizations for the allocation and marking procedures.

The result of the FAC benchmark[2], given in table 1, indicate that both algorithms have a comparable performance.

3 Embedding into the EuLisp Runtime System

Our compiler is written in an EuLisp-like syntax using an "EuLisp compatibility package" defined on top of CLtL2 [Steele Jr., 1990]. The basic part of the EuLisp run time system is written in TAIL[3] which allows the handling of C-like data structures and contains a foreign function interface. The interface functions of the memory management system are declared as foreign functions in TAIL. Except of instances of the class <fixed-precision-integer> all instances of EuLisp classes are represented as pointer to data. EuLisp basic classes are described similarly to the following definition of the class <cons>:

```
(%define-standard-class
  (<cons> <class>);;class and metaclass
        <list>               ;;superclass
        ((car type <object>   ;;car slot holds objects with type <object>
              reader car      ;;generate reader car
              writer set-car) ;;generate writer set-car
         (cdr type <object>
              reader cdr
              writer set-cdr ))
        constructor (cons car cdr);;generate function cons
        allocation single-card    ;;allocate instances on STSS cards
        representation pointer-to-struct);; representation is pointer
                                         ;; to a structure with 2 slots
```

The last specifications describe how instances should be allocated. Init-forms for the initialization of the memory management system and if specified constructor as well as slot accessor functions for instances are generated automatically by the compiler. In addition, depending on the type that slots could hold and dependent on the specified representation a specialized tracing function for each class is generated.

When ever possible, data are allocated statically during compilation time. Statically allocated data are represented with their class in front as on MTSS-cards .

[2] To allow a comparison with the B/W-algorithm, no user-specified roots were used.

[3] TAIL, an acronym from Typed Implementation Language, was developed in the framework of the project Apply.

4 Some Performance Tests with EuLisp Applications

This section shows the results of two benchmarks, a cons-intensive version of reverse[4]

```
(defun reverse (l)
  (if (null l) ()
      (if (null (cdr l))
        l
        (cons (car (reverse (cdr l)))
              (reverse (cons (car l)
                             (reverse (cdr (reverse (cdr l))))))))))
```

that uses only the functions cons, car and cdr but no auxiliary functions, and the traverse benchmark [Gabriel, 1986]. The corresponding EuLisp modules were compiled into ANSI-C-code and then compiled and linked with the memory management system using the GNU C compiler/linker. The allocator is configured to start with 16 cards of size 4096 bytes. If after a garbage collection more than one third of the allocated heap is still in use then the heap size will be doubled. The root set consists of all statically allocated EuLisp data contained in a certain "data" section and of the activation stack. To be more realistic we use EuLisp I/O to force the allocation of instances of classes like <string> , <stream> etc., which do not appear in the actual benchmark programs. These instances are allocated on all three types of cards. Therefore, no special optimization of the allocation procedures can be done as in the case of the previous FAC benchmark. The result of the reverse benchmark[5] is given in table 2 for conses allocated on STSS, MTSS and STMS cards, respectively. For comparison, the values of these benchmarks obtained with Franz Allegro 4.1 using the compiler options (optimize (speed 3) (safety 0)) are given too. Many Lisp systems use special optimizations for conses. Thus the table also shows the benchmark for ACL4.1 where conses are replaced by corresponding unnamed structures of type list. The results prove satisfactory, the best result is obtained for the allocation on STSS cards. Using a configuration with a larger number of initial cards the performance can be further improved. That is shown in table 3. The numbers presented indicate that there is an optimum at about 256 cards where user time is small and system time is at its minimum, whereas for larger numbers system time begins to increase more strongly than user time decreases.[6] The results of the traverse benchmarks are given in table 4. Here the instances of the structure <node> are allocated on MTSS cards.

For a final conclusion we have to to perform other and more complicated benchmarks. However, the results obtained until now indicate that our modified conservative memory management scheme is suitable at least for small stand alone Lisp application.

[4] Reversing a list with 14 elements requires the allocation of 4427042 conses.

[5] All measurements are performed on a Sun ELC with 32 Mbyte and Apply EuLisp-¿C compiler version 93-09.

[6] For a version of reverse which uses standard C malloc without any garbage collection we measured a system time of about 20 s.

Performance in dependence on card types				
Test	Boehm/Weiser	STSS	MTSS	STMS
FAC	11.3 s	9.4 s	10.1 s	11.4 s

Table 1: Factorial bencmark in C

REVERSE Benchmark I		
	time	
System	user	system
EuLisp STSS	21.1 ± 0.1 s	1.2 ± 0.1 s
EuLisp MTSS	26.7 ± 0.3 s	1.9 ± 0.3 s
EuLisp STMS	33.4 ± 0.2 s	2.0 ± 0.1 s
ACL4.1 with cons	12.4 ± 0.0 s	0.1 ± 0.1 s
ACL4.1 with struct	61.1 ± 0.04 s	0.05 ± 0.02 s

Table 2: Reverse benchmark for conses allocated on different types of cards

REVERSE Benchmark II									
	STSS			MTSS			STMS		
		time			time			time	
Cards	gc	user	system	gc	user	system	gc	user	system
16	927	21.1 s	1.2 s	1406	26.7 s	1.9 s	1406	33.4 s	2.0 s
32	350	17.7 s	0.5 s	527	21.9 s	0.7 s	527	30.6 s	0.8 s
64	155	16.4 s	0.2 s	234	20.0 s	0.3 s	234	29.5	0.4 s
128	73	15.8 s	0.2 s	110	19.1 s	0.2 s	110	28.9 s	0.2 s
256	36	15.5 s	0.2 s	54	18.8 s	0.2 s	54	28.8 s	0.2 s
512	17	15.4 s	0.3 s	26	18.6 s	0.3 s	26	28.4 s	0.3 s
1024	8	15.4 s	0.5 s	13	18.5 s	0.6 s	13	28.5 s	0.6 s

Table 3: Reverse benchmark for conses allocated on different types of cards in dependece of the number of initial cards

TRAVERSE-INITIALIZE Benchmark		
System	user time	system time
EuLisp	2.0 ± 0.01 s	0.06 ± 0.02
ACL4.1 with cons	0.5 s	0.02 s
TRAVERSE-RUN Benchmark		
System	user time	system time
EuLisp	4.93 ± 0.01 s	0 ± 0.01 s
ACL4.1 with cons	4.65 s	0.01 ± 0.01 s

Table 4: Results of the Traverse-Run Benchmark

References

[Boehm and Weiser, 1988] H.-J. Boehm and M. Weiser. Garbage Collection in an Uncooperative Environment. *Software - Practice and Experience*, 18(9):807–820, 1988.

[Bretthauer *et al.*, 1992] H. Bretthauer, T. Christaller, H. Friedrich, W. Goerigk, W. Heicking, U. Hoffmann, D. Hovekamp, H. Knutzen, J. Kopp, E. U. Kriegel, I. Mohr, R. Rosenmüller, and F. Simon. APPLY: A modern and practical Lisp. APPLY-Arbeitspapier APPLY/GMD/XIII/6, CAU/GMD/ISST/VW-GEDAS, Sankt Augustin, September 1992.

[Gabriel, 1986] R. P. Gabriel. *Performance and Evaluation of Lisp Systems.* MIT Press, Cambridge, Massachusetts, 1986.

[Padget *et al.*, 1993] J. Padget, G. Nuyens, and H. Bretthauer. An Overview of EuLisp. APPLY-Arbeitspapier APPLY/GMD/II/1, Gesellschaft für Mathematik und Datenverarbeitung (GMD), Sankt Augustin, May 1993.

[Steele Jr., 1990] G. L. Steele Jr. *Common Lisp - The Language, Second Edition.* Digital Press, Bedford, Massachusetts, 1990.