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

E. Ulrich Kriegel

Fraunhofer Institute for Software Engineering and Systems Engineering

Task: Kernel
Status: Final version
Document Identifier: APPLY/ISST/V6.1/4
Created: 07/01/1993
Revised: 14/10/1993
Synopsis

This paper is identical to that one contained in the collection of papers of the OOPSLA’93 Workshop on Garbage Collection and Memory Management, Washington DC, September 27, 1993. It describes the utilization of an conservative garbage collection scheme in an Eulisp to ASM/C compiler.

This work was supported by the German Federal Ministry for Research and Technology (BMFT) within the joint project APPLY. The partners in this project are the Christian Albrechts University Kiel, the Fraunhofer Institute for Software Engineering and Systems Engineering (ISST), the German National Research Center for Computer Science (GMD), and VW-Gedas.

Christian Albrechts University Kiel
Institute for Computer Science and Practical Mathematics
Research Group “Programming Languages and Compiler Construction”
Preussenerstr. 1-9, D-24105 Kiel, FRG
Fax: 0431/5661-43
Person to contact: Dr. Friedemann Simon
Phone: 0431/5604-25
E-mail: fs@informatik.uni-kiel.de

Fraunhofer Institute for Software Engineering and Systems Engineering (ISST)
Kurstr. 33, D-10117 Berlin, FRG
Fax: 030/20372-301
Person to contact: Dr. Horst Friedrich
Phone: 030/20372-342
E-mail: horst.friedrich@isst.fhg.de

German National Research Center for Computer Science (GMD)
Institute for Applied Information Technology
Artificial Intelligence Research Division
P.O. Box 1316, D-53757 Sankt Augustin, FRG
Fax: 02241/14-2072
Person to contact: Prof. Dr. Thomas Christaller
Phone: 02241/14-2678
E-mail: thomas.christaller@gmd.de

VW-GEDAS
Pascalstr. 11, D-10587 Berlin, FRG
Fax: 030/3900799
Person to contact: Dipl.-Ing. Jürgen Beste
Phone: 030/39007-331
E-mail: beste%sun1@hwsw.gedas.de
Contents

1 Introduction 1

2 Layout of the Memory Management Scheme 1

3 Embedding into the EuLisp Runtime System 3

4 Some Performance Tests with EuLisp Applications 3
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 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\textsuperscript{1} to carry out performance tests. Compared to the original algorithm memory

\textsuperscript{1}We used a version of factorial using an arithmetic for natural numbers where each number $N$ is represented as the $N$th. successor of zero.
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.

  **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.

---

\(^2\)To allow a comparison with the B/W-algorithm, no user-specified roots were used.
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 ChtL2 [Steele Jr., 1990]. The basic part of the EuLisp runtime 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>:

```lisp
(%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.

4 Some Performance Tests with EuLisp Applications

This section shows the results of two benchmarks, a cons-intensive version of reverse\(^4\)

```lisp
(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)))))))))
```

\(^3\)TAIL, an acronym from Typed Implementation Language, was developed in the framework of the project Apply.

\(^4\)Reversing a list with 14 elements requires the allocation of 4427042 conses.
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 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. 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.

---

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 100 s.
### Table 1: Factorial benchmark in C

<table>
<thead>
<tr>
<th>System</th>
<th>user</th>
<th>system</th>
</tr>
</thead>
<tbody>
<tr>
<td>EuLisp STSS</td>
<td>21.1 ± 0.1 s</td>
<td>1.2 ± 0.1 s</td>
</tr>
<tr>
<td>EuLisp MTSS</td>
<td>26.7 ± 0.3 s</td>
<td>1.9 ± 0.3 s</td>
</tr>
<tr>
<td>EuLisp STMS</td>
<td>33.4 ± 0.2 s</td>
<td>2.0 ± 0.1 s</td>
</tr>
<tr>
<td>ACL4.1 with cons</td>
<td>12.4 ± 0.0 s</td>
<td>0.1 ± 0.1 s</td>
</tr>
<tr>
<td>ACL4.1 with struct</td>
<td>61.1 ± 0.04 s</td>
<td>0.05 ± 0.02 s</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Cards</th>
<th>gc</th>
<th>user</th>
<th>system</th>
<th>gc</th>
<th>user</th>
<th>system</th>
<th>gc</th>
<th>user</th>
<th>system</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>927</td>
<td>21.1 s</td>
<td>1.2 s</td>
<td>1406</td>
<td>26.7 s</td>
<td>1.9 s</td>
<td>1406</td>
<td>33.4 s</td>
<td>2.0 s</td>
</tr>
<tr>
<td>32</td>
<td>350</td>
<td>17.7 s</td>
<td>0.5 s</td>
<td>527</td>
<td>21.9 s</td>
<td>0.7 s</td>
<td>527</td>
<td>30.6 s</td>
<td>0.8 s</td>
</tr>
<tr>
<td>64</td>
<td>155</td>
<td>16.4 s</td>
<td>0.2 s</td>
<td>234</td>
<td>20.0 s</td>
<td>0.3 s</td>
<td>234</td>
<td>29.5 s</td>
<td>0.4 s</td>
</tr>
<tr>
<td>128</td>
<td>73</td>
<td>15.8 s</td>
<td>0.2 s</td>
<td>110</td>
<td>19.1 s</td>
<td>0.2 s</td>
<td>110</td>
<td>28.9 s</td>
<td>0.2 s</td>
</tr>
<tr>
<td>256</td>
<td>36</td>
<td>15.5 s</td>
<td>0.2 s</td>
<td>54</td>
<td>18.8 s</td>
<td>0.2 s</td>
<td>54</td>
<td>28.8 s</td>
<td>0.2 s</td>
</tr>
<tr>
<td>512</td>
<td>17</td>
<td>15.4 s</td>
<td>0.3 s</td>
<td>26</td>
<td>18.6 s</td>
<td>0.3 s</td>
<td>26</td>
<td>28.4 s</td>
<td>0.3 s</td>
</tr>
<tr>
<td>1024</td>
<td>8</td>
<td>15.4 s</td>
<td>0.5 s</td>
<td>13</td>
<td>18.5 s</td>
<td>0.6 s</td>
<td>13</td>
<td>28.5 s</td>
<td>0.6 s</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>System</th>
<th>user</th>
<th>system time</th>
</tr>
</thead>
<tbody>
<tr>
<td>EuLisp</td>
<td>2.0 ± 0.01 s</td>
<td>0.06 ± 0.02 s</td>
</tr>
<tr>
<td>ACL4.1 with cons</td>
<td>0.5 s</td>
<td>0.02 s</td>
</tr>
</tbody>
</table>

### Table 4: Results of the Traverse-Run Benchmark

<table>
<thead>
<tr>
<th>System</th>
<th>user</th>
<th>system time</th>
</tr>
</thead>
<tbody>
<tr>
<td>EuLisp</td>
<td>4.93 ± 0.01 s</td>
<td>0 ± 0.01 s</td>
</tr>
<tr>
<td>ACL4.1 with cons</td>
<td>4.65 s</td>
<td>0.01 ± 0.01 s</td>
</tr>
</tbody>
</table>
References


