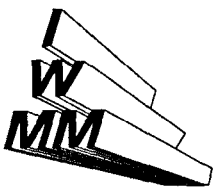
Lecture Notes in Computer Science 986
Edited by G. Goos, J. Hartmanis and J. van Leeuwen

Advisory Board: W. Brauer D. Gries J. Stoer

Henry G. Baker (Ed.)

Memory Management

International Workshop IWMM 95
Kinross, UK, September 27-29, 1995
Proceedings



Springer

Series Editors

Gerhard Goos
Universität Karlsruhe
Vincenz-Priessnitz-Straße 3, D-76128 Karlsruhe, Germany

Juris Hartmanis
Department of Computer Science, Cornell University
4130 Upson Hall, Ithaca, NY 14853, USA

Jan van Leeuwen
Department of Computer Science,Utrecht University
Padualaan 14, 3584 CH Utrecht,The Netherlands

Volume Editor

Henry G. Baker
Synapse Computer Services
16231 Meadow Ridge Way, Encino 91436, CA, USA

Cataloging-in-Publication data applied for

Die Deutsche Bibliothek - CIP-Einheitsaufnahme

Memory management : international workshop ; proceedings / IWMM 95, Kinross, UK, September 1995 / Henry G. Baker (ed.). - Berlin ; Heidelberg ; New York : Springer, 1995
(Lecture notes in computer science ; Vol. 986)
ISBN 3-540-60368-9
NE: Baker, Henry G. [Hrsg.]; IWMM <1995, Kinross>; GT

CR Subject Classification (1991): D.4.2, B.3.2, B.5.1, B.6.1, B.7.1, C.1.2, D.4.7-8, D.1, D.3.2

ISBN 3-540-60368-9 Springer-Verlag Berlin Heidelberg New York

Typesetting: Camera-ready by author
SPIN 10485579 06/3142 – 5 4 3 2 1 0 Printed on acid-free paper

IWMM'95 Call For Papers

Memory management of dynamically allocated memory (MM) plays a large and increasingly important part in the interface between sophisticated languages (Lisp, Scheme, ML, Prolog, Smalltalk, Modula-3, Eiffel, constraint languages, etc.) and operating systems. MM interacts with real-time scheduling, concurrency control, parallel threads, persistent objects, distributed objects, active objects, orphan elimination, finalization, multi-lingual environments, etc.

Advances in memory devices (speed, size, power, access characteristics, compression) and the demands of new applications (e.g., desktop audio/video, distributed databases/applications on high performance, low-latency networks) provide additional problems and opportunities in MM.

IWMM'92 was a highly successful workshop which brought together researchers and practitioners working on various aspects of MM. IWMM'95 is intended to keep the same wide-ranging and eclectic scope to promote the cross-fertilization that was achieved by IWMM'92. In addition to a mix of theoretical and practical papers, we are also seeking papers with interdisciplinary and/or pioneering content.

Topics of Interest

Explicit alloc/free algorithms/measurements
Garbage Collection (GC)
Parallel/real-time GC
Multilingual GC
Environment structures
Static/Dynamic MM
Backtracking and MM
Constraints and MM
MM for parallel languages
MM and memory hierarchies
Precaching strategies and MM
Compile time GC
Definition/minimization of storage leaks
MM of persistent objects
MM of distributed objects
Architecture/OS support for MM
MM and distributed shared memory
Hardware support for MM&GC
MM performance analysis & optimization tools
Reflective MM

Preface

The International Workshop on Memory Management 1995 (IWMM'95) is a continuation of the excellent series started by Yves Bekkers and Jacques Cohen with IWMM'92. The present volume assembles the refereed and invited technical papers which were presented during this year's workshop. The Call For Papers for IWMM'95 is included as an appendix to this Preface.

Memory storage space is as fundamental to computing as the time for CPU cycles, as was shown by Alan Turing. An expanding amount of time is not very useful for more complex computations without a corresponding expanding amount of space. We can make an analogy between storage space in a computer memory and energy in a physical system. A physical system with a limited amount of energy is like a child's wind-up toy — it can express only a limited range of behavior before running down. Similarly, a computer program with only a limited access to memory space also exhibits a very limited range of behavior.

Later studies of automata models of computers have shown that while an expanding amount of space is necessary for interesting behavior, it is not sufficient. If the access to the memory is constrained to occur in certain orders — for example, in only a stack-like (Last-In, First-Out or LIFO) order — then the range of behavior is also constrained.

It is therefore no accident that *progress in the field of computer software can be directly correlated with the removal of limitations on the amount and access patterns of computer memory.* The static memory models of Fortran and Cobol have given way to the stack memory models of Algol-60, Pascal and C. And these stack models have given way to the randomly-accessed heap storage of Lisp, Algol-68, Prolog, Eiffel and finally C++.

Improvements in the exploitation of computer memory have come very slowly, with strong rearguard actions required. One of the reasons for this conservatism is that serious constraints on memory management were built into the fundamentals of popular computer languages, and progress has therefore required the adoption of new computer languages. Each tiny step has required a transition whose pain is essentially independent of the size of the step. Another reason for conservatism is the fact that the conceptual gap between the models of computation useful for software development and the models of computation used for computer hardware has continued to widen. Today, there is a vast gulf between the dynamic random-access memory chip (DRAM) provided by the hardware designer and the dynamic object-oriented graph structure desired by the software designer. This gulf must be filled by memory management hardware and software.

The object model of computation was pioneered in the language Lisp. In the object model of computation, the computer memory consists of a dynamic set of objects, each of which may "point to" zero or more other

objects. Some objects are accessible directly via anchors or "roots", while others are accessible indirectly by following chains of pointers from one object to another. Thus, the object model is a dynamic "directed graph" structure, in which additional vertices ("nodes" or objects) may be added or removed, and in which edges may be dynamically redirected from one object to another.

This dynamic object graph model was a dramatic improvement over the linear tape storage of Turing or the linear RAM of von Neumann. However, because the object model is so different from these memory hardware concepts, a non-trivial layer of mechanism is required to provide "objects" in a memory designed only for array elements. This layer involves the dynamic allocation of small contiguous chunks of the linear RAM "address space", and the installation and maintenance of pointers among these chunks of memory.

Various schemes for dynamic memory allocation were tried, and the basic schemes we use today were developed by 1960. "List memory" was developed in Newell, Shaw and Simon's IPL language, "reference counting" was developed by Collins for a computer algebra system, and "mark/sweep garbage collection" was developed by McCarthy for the Lisp language. (At the same time, the far less capable systems of static and stack storage were being utilized in the Fortran, Cobol and Algol-60 languages.)

As these systems of dynamic storage management were developed, their flaws also became clear. Newell, Shaw and Simon found in the mid-1950s that programmer-directed object deletion was buggy and unworkable in a system of any complexity — a lesson that can apparently only be learned the hard way at the "school of hard knocks," if the subsequent history of computer languages which tried to cut this corner — e.g., Pascal, C, C++ — is any guide.

The alternative to programmer-directed deletion of objects which are no longer useful is "automatic" memory management, in which the memory manager itself recovers the storage from useless objects. Two classical techniques for automatic memory management are *reference counting* and *marking garbage collection.*

Collins found that reference counting is useful in certain contexts, but has an overhead which is proportional to the length of the computation, and is not capable of detecting cycles of useless objects.

McCarthy found that mark/sweep garbage collection (GC) can collect such cycles of "garbage", and thus is essentially the only method of memory management that can handle a general object-oriented directed graph structure. Furthermore, he also found that marking GC is very efficient with larger memory sizes because the fixed amount of work of marking for a set of live objects can be amortized over the larger number of garbage cells that are collected in one sweep of these larger memories. Unfortunately, McCarthy's implementation of mark/sweep GC required that the application program completely stop dead in its tracks while the garbage collection process was going on. While various proposals were made to solve this problem, unfor-

tunately none was implemented before the computing world split into the faction that advocated garbage collection and worked on non-time-critical problems, and the faction that felt that garbage collection was perhaps too complex and too difficult for more time-critical problems.

There are good reasons for the difficulty of the general task of managing object-oriented memory. The primary reason for using the object model is to allow for the controlled sharing of information among objects. This sharing, however, so blurs the boundaries of "ownership" of chunks of memory that no isolated object or application can "see" enough of the object graph to know whether a particular object is useless. Thus, the desire for the advantages of sharing *causes* the desire for automatic memory management. Furthermore, the more complex the sharing patterns, the more difficult the management problem. For example, so long as the sharing pattern is acyclic, thus representing essentially *finite* structures, reference counting is adequate. If, however, *infinite* structures must be represented by means of directed cycles, then reference counting is inadequate, and more general marking garbage collection is required.

It is now 1995, however, and several computing chickens have come home to roost. "Objects" have now taken over the computing world, so the efficient management of memory to provide for the storage of these objects has become a serious concern. The costs of software development continue to escalate, so that memory management techniques that can remove burdens from the programmer are of great interest. The "central processing" part of the computer CPU has been sped up to the point where the bottleneck in application processing speed is no longer arithmetic, but memory (hierarchy) management. The promised speedups from "parallel processing" on a dedicated parallel processor have never materialized for most applications, while the requirements for "distributed" processing have become quite insistent.

The original revulsion against the complexity of marking garbage collection has now evolved into an admiration of the elegance of a simple straightforward idea that can replace a myriad of buggy *ad hoc* hacks that still cannot reliably collect all the useless objects in an object-oriented system.

Real-time marking garbage collection algorithms have been developed which no longer require that an application stop for long periods during garbage collection. Indeed, hardware-assisted garbage collectors can provide guaranteed access times which are little different from those of memory modules which do not provide garbage collection.

For a serial computer, the consensus is that some form of marking garbage collection is now the technique of choice, having displaced reference counting as too expensive (due to the expense of count maintenance) and too restrictive (due to its inability to collect cycles of garbage), and programmer-directed reclamation as too buggy, too dangerous, and too unreliable.

Significant additional research is required, however, to allow the garbage collector to work more closely with the compiler, the operating system, and the memory hierarchy.

The popularity of the World Wide Web on the Internet has focused attention on the need for efficient, robust methods for "distributed" memory management, in which portions of an application are spread out on various machines at disparate locations on a network.

Distributed garbage collection has turned out to be a quite difficult problem. The combined requirements of handling faulty communications links, faulty processors, and faulty software, together with the usual locking and synchronization problems of concurrent systems, have so far hindered researchers from producing efficient, robust distributed garbage collectors.

Once again, there are good reasons for the difficulty of distributed garbage collection. Before an object can be collected, all processors in the distributed system must *agree* that the object is useless, and it has been found that reaching such a consensus in the presence of various kinds of failures is very difficult and sometimes impossible.

In conclusion, we find that effective memory management — including sophisticated marking garbage collection techniques — is a fundamental building block in reliable and efficient computer languages — including those intended for real-time and distributed applications.

We wish to thank the authors for submitting their papers, and the referees for their careful evaluation of the submitted papers, and their suggestions to the authors for improving their papers. We also thank Peter Dickman for his help in hosting this workshop, as well as Eric Jul and Michael Svendsen for their handling of the electronic submissions. We also wish to thank the Cambridge Research Laboratory of Apple Computer, Inc. for their help in hosting the IWMM'95 Program Committee Meeting.

Henry G. Baker
Program Chair, IWMM'95
Encino, CA, USA
hbaker@netcom.com
July 1995

Correctness and analysis
Laziness and MM

Program committee

Henry Baker, Chair
Yves Bekkers, IRISA, France
Hans-Jurgen Boehm, Xerox PARC, USA
Jacques Cohen, Brandeis University, USA
Bart Demoen, K. U. Leuven, Belgium
Peter Dickman, University of Glasgow, UK
Benjamin Goldberg, New York University, USA
Eric Jul, DIKU, Denmark
David Moon, Apple Computer, USA
Dan Sahlin, SICS, Sweden
Paul Wilson, University of Texas, USA
Taiichi Yuasa, Toyohashi University, Japan

Local Arrangements

Peter Dickman, University of Glasgow, UK, with assistance from the Department of Computing Science, University of Glasgow

Publicity and Communications

Eric Jul, DIKU, University of Copenhagen, Denmark, with assistance from DIKU

Table of Contents