CHAPTER 7

CONCLUDING REMARKS

In this study, we have explored a heretofore neglected alternative to generational garbage collection and found it to be competitive with generational algorithms, and in many cases superior to them. We introduced a simple new algorithm, called deferred older-first, within the classification of age-based algorithms. We described an efficient implementation of the new algorithm, its heap organization, copying mechanisms, and pointer-maintenance mechanisms (Chapter 3). We demonstrated, in a comparative trace-based simulation study, that the new algorithm achieves a significantly lower copying cost than generational collection on many programs, sometimes as much as 11 times lower (Chapter 5). We showed that, even though the pointer-maintenance cost of the new algorithm is usually higher than that of generational collection, the total cost is nevertheless lower on many programs, sometimes as much as 4 times (Chapter 6). These results were unexpected in light of the prior understanding of generational collector performance, which suggested that collecting among the youngest data achieves the best performance.

Using profiling of an age-ordered heap, our investigation provides insight into the copying performance of the new algorithm, and shows how it genuinely concentrates effort on those object ages, neither the youngest nor the oldest in the heap, which result in the lowest copying cost. Our new algorithm is very simple, and non-adaptive, yet its performance is good. While much work has been devoted to the design of adaptive generational algorithms, adaptive versions of our alternative, or indeed, other alternatives, are unexplored, but, according to our results, promising. Thus the field of garbage collection, even in the simplest setting of stop-and-copy uniprocessor algorithms, is reopened for study.

In our examination of the pointer-maintenance structures to support the demands of new algorithms, we found that precepts for the design of pointer maintenance from existing studies of generational collection do not carry over to alternative algorithms (for instance, the number of duplicate remembered set insertions is much greater in generational collection), and thus pointer maintenance design is also open for further study. Furthermore, the investigation of the positional distribution of pointers dispells the oft-cited notion that younger-to-older pointers dominate even in object-oriented languages.

This work has exposed the facts about the performance of copying garbage collection algorithms, the factors affecting it, and the limits of its possible improvement. However, the settings in which the experimental results were obtained were necessarily limited, and other settings remain to be explored.

We used trace-based simulation, albeit in conjunction with a prototype implementation of the collectors. However, traces do not capture stack processing costs nor the interactions between the collector and the mutator. A desirable follow-up study is to integrate the several collectors we described with a running mutator system, such as a Java virtual machine. This integration will permit the verification of the design we outlined for implementation, the validation of the cost models we assumed (Chapter 3), the measurement of memory locality effects, and the final verdict in the form of absolute timing of different algorithms.

In this study, we eschewed analytical modelling of garbage collection costs. Modelling the actions of a collector presupposes the availability of analytical models for the behavior of the mutator. Whereas some models have been attempted for object lifetimes, with unclear utility, models for pointer structure are entirely unexplored. The success of the deferred older-first collector as an approximation of the ideal queue collector (Chapter 1), the heap profiles (Chapter 5), and the pointer distributions (Chapter 6) indicate a high degree of spatio-temporal correlation of object lifetimes and of pointer structure, suggesting that first-order statistical models will be inadequate.

Our investigation focused on the application of collectors to the entire heap, from the most recently allocated objects to the oldest ones. As we discovered in the study of demise positions as well as of pointer directions, and entirely in accordance with intuitive expectations, an advantage is possible in large and long-running systems if an *a priori* division is imposed on the heap, in which the most recently allocated data are managed separately to lower the pointer maintenance overhead for stores into new objects, the oldest data are identified as permanent (by programmer declaration, or adaptively) to exclude them from consideration entirely, and the middle portion, representing an active mature space, is managed by a deferred-older-first algorithm. Indeed, this observation can be seen as the basis for defining policies on top of the mechanisms described for the Mature Object Space and its *train* algorithm [Hudson and Moss, 1992; Seligmann and Grarup, 1995].

The evaluation of different algorithms applied to a mature space could, in principle, proceed as in this study. The trace of mature space objects can be obtained either as the sequence of objects promoted out of a concrete younger-space collector, or, more abstractly, as the subsequence of objects with lifetimes above a fixed threshold extracted from a full sequence of allocated objects. Since the volume of objects presented to the mature space collector is, by design, much smaller than the total volume of allocation, then, in order for the mature space collector to be exercised sufficiently (Section 4.2.2.2) the total allocation must be considerably larger. We believe that it will prove infeasible to obtain accurate traces as for this study, and that direct implementation and measurement will be especially needed for the mature space.

BIBLIOGRAPHY

[Abadi and Cardelli, 1996] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, New York, 1996.

[Aditya *et al.*, 1994] Shail Aditya, Christine Flood, and James Hicks. Garbage collection for strongly-typed languages using run-time type reconstruction. In *Proceedings of SIGPLAN'94 Conference on Programming Languages Design and Implementation*, volume 29 of *SIGPLAN Notices*, pages 12–23, Orlando, Florida, June 1994. ACM Press. Also Lisp Pointers VIII 3, July–September 1994.

[Agesen *et al.*, 1998] Ole Agesen, David Detlefs, and J. Eliot B. Moss. Garbage collection and local variable type-precision and liveness in Java(TM) virtual machines. In *Proceedings of SIGPLAN'98 Conference on Programming Languages Design and Implementation*, volume 33 of *SIGPLAN Notices*, pages 269–279, Montreal, Québec, Canada, June 1998. ACM Press.

[Appel and Shao, 1996] Andrew W. Appel and Zhong Shao. Empirical and analytic study of stack versus heap cost for languages with closures. *Journal of Functional Programming*, 6(1):47–74, January 1996.

[Appel, 1987] Andrew W. Appel. Garbage collection can be faster than stack allocation. *Information Processing Letters*, 25(4):275–279, 1987.

[Appel, 1989a] Andrew W. Appel. Runtime tags aren't necessary. *Lisp and Symbolic Computation*, 2:153–162, 1989.

[Appel, 1989b] Andrew W. Appel. Simple generational garbage collection and fast allocation. *Software Practice and Experience*, 19(2):171–183, 1989.

[Appel, 1992] Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, Cambridge, 1992.

[Appel, 1997] Andrew Appel. A better analytical model for the strong generational hypothesis. November 1997.

[Baker, 1978] Henry G. Baker. List processing in real-time on a serial computer. *Communications of the ACM*, 21(4):280–94, 1978. Also AI Laboratory Working Paper 139, 1977.

[Baker, 1990] Henry G. Baker. Unify and conquer (garbage, updating, aliasing, ...) in functional languages. In LFP [1990], pages 218–226.

[Baker, 1993] Henry G. Baker. 'Infant Mortality' and generational garbage collection. *SIGPLAN Notices*, 28(4):55–57, 1993.

[Barendregt, 1984] Henk P. Barendregt. *The Lambda Calculus: Its Syntax and Semantics*. Elsevier (North-Holland), Amsterdam, 1984. Second edition.

[Barrett and Zorn, 1995] David A. Barrett and Benjamin Zorn. Garbage collection using a dynamic threatening boundary. In *Proceedings of SIGPLAN'95 Conference on Programming Languages Design and Implementation*, volume 30 of *SIGPLAN Notices*, pages 301–314, La Jolla, CA, June 1995. ACM Press.

[Bekkers and Cohen, 1992] Yves Bekkers and Jacques Cohen, editors. *International Workshop on Memory Management*, number 637 in Lecture Notes in Computer Science, St. Malo, France, September 1992. Springer-Verlag.

[Bekkers *et al.*, 1986] Yves Bekkers, B. Canet, Olivier Ridoux, and L. Ungaro. MALI: A memory with a real-time garbage collector for implementing logic programming languages. In *3rd Symposium on Logic Programming*, pages 258–264. IEEE Press, 1986.

[Bishop, 1977] Peter B. Bishop. *Computer Systems with a Very Large Address Space and Garbage Collection*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1977.

[Boehm and Shao, 1993] Hans-Juergen Boehm and Zhong Shao. Inferring type maps during garbage collection. In Moss et al. [1993].

[Boehm and Weiser, 1988] Hans-Juergen Boehm and Mark Weiser. Garbage collection in an uncooperative environment. *Software Practice and Experience*, 18(9):807–820, 1988.

[Boehm, 1993] Hans-Juergen Boehm. Space efficient conservative garbage collection. In *Proceedings of SIGPLAN'93 Conference on Programming Languages Design and Implementation*, volume 28(6) of *SIGPLAN Notices*, pages 197–206, Albuquerque, New Mexico, June 1993. ACM Press.

[Caudill and Wirfs-Brock, 1986] Patrick J. Caudill and Allen Wirfs-Brock. A third generation Smalltalk-80 implementation. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 119–130, Portland, Oregon, September 1986. *SIGPLAN Notices 21*, 11 (November 1986).

[Chambers, 1992] Craig Chambers. *The Design and Implementation of the SELF Compiler, an Optimizing Compiler for an Objected-Oriented Programming Language*. PhD thesis, Stanford University, March 1992.

[Cheney, 1970] C. J. Cheney. A non-recursive list compacting algorithm. *Communications of the ACM*, 13(11):677–8, November 1970.

[Clinger and Hansen, 1997] William D. Clinger and Lars T. Hansen. Generational garbage collection and the radioactive decay model. *SIGPLAN Notices*, 32(5):97–108, May 1997. *Proceedings of the ACM SIGPLAN '97 Conference on Programming Language Design and Implementation*.

[Cohen, 1981] Jacques Cohen. Garbage collection of linked data structures. *Computing Surveys*, 13(3):341–367, September 1981.

[Courts, 1988] Robert Courts. Improving locality of reference in a garbage-collecting memory management system. *Communications of the ACM*, 31(9):1128–1138, September 1988.

[Cox and Oakes, 1984] D. R. Cox and D. Oakes. *Analysis of Survival Data*. Chapman and Hall, London, 1984.

[Demers *et al.*, 1990] Alan Demers, Mark Weiser, Barry Hayes, Daniel G. Bobrow, and Scott Shenker. Combining generational and conservative garbage collection: Framework and implementations. In *Conference Record of the Seventeenth Annual ACM Symposium on Principles of Programming Languages*, SIGPLAN Notices, pages 261–269, San Francisco, CA, January 1990. ACM Press.

[DeTreville, 1990a] John DeTreville. Experience with concurrent garbage collectors for Modula-2+. Technical Report 64, DEC Systems Research Center, Palo Alto, CA, August 1990.

[DeTreville, 1990b] John DeTreville. Heap usage in the Topaz environment. Technical Report 63, DEC Systems Research Center, Palo Alto, CA, August 1990.

[Diwan *et al.*, 1992] Amer Diwan, J. Eliot B. Moss, and Richard L. Hudson. Compiler support for garbage collection in a statically typed language. In *Proceedings of SIGPLAN'92 Conference on Programming Languages Design and Implementation*, volume 27 of *SIGPLAN Notices*, pages 273–282, San Francisco, CA, June 1992. ACM Press.

[Diwan *et al.*, 1995] Amer Diwan, David Tarditi, and J. Eliot B. Moss. Memory system performance of programs with intensive heap allocation. *ACM Transactions on Computer Systems*, 13(3):244–273, August 1995.

[Elandt-Johnson and Johnson, 1980] Regina C. Elandt-Johnson and Norman Lloyd Johnson. *Survival Models and Data Analysis*. Wiley, New York, 1980.

[Fenichel and Yochelson, 1969] Robert R. Fenichel and Jerome C. Yochelson. A Lisp garbage collector for virtual memory computer systems. *Communications of the ACM*, 12(11):611–612, November 1969.

[Fradet, 1994] Pascal Fradet. Collecting more garbage. In LFP [1994].

[Goldberg and Gloger, 1992] Benjamin Goldberg and Michael Gloger. Polymorphic type reconstruction for garbage collection without tags. In *Conference Record of the 1992 ACM Symposium on Lisp and Functional Programming*, pages 53–65, San Francisco, CA, June 1992. ACM Press.

[Goldberg and Robson, 1983] Adele Goldberg and David Robson. *Smalltalk-80: The Language and its Implementation*. Addison-Wesley, 1983.

[Gonçalves, 1995] Marcelo J. R. Gonçalves. *Cache Performance of Programs with Intensive Heap Allocation and Generational Garbage Collection*. PhD thesis, Princeton University, May 1995.

[Gosling *et al.*, 1996] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, Reading, Mass., 1996.

[Grarup and Seligmann, 1993] Steffen Grarup and Jacob Seligmann. Incremental mature garbage collection. Technical report, Computer Science Department, Aarhus University, Aarhus, Denmark, August 1993. M.Sc. Thesis.

[Harrison, 1989] Williams Ludwell Harrison, III. The Interprocedural Analysis and Automatic Parallelization of Scheme Programs. *Lisp and Symbolic Computation*, 2(3/4):179–396, October 1989.

[Hayes, 1991] Barry Hayes. Using key object opportunism to collect old objects. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 33–46, Phoenix, Arizona, October 1991. *SIGPLAN Notices 26*, 11 (November 1991).

[Hayes, 1993] Barry Hayes. *Key Objects in Garbage Collection*. PhD thesis, Stanford University, Stanford, California, March 1993.

[Hicks *et al.*, 1998] Michael Hicks, Luke Hornof, Jonathan T. Moore, and Scott M. Nettles. A study of large object spaces. In *International Symposium on Memory Management*, pages 138–145, Vancouver, British Columbia, Canada, October 1998. ACM Press.

[Hosking and Hudson, 1993] Antony L. Hosking and Richard L. Hudson. Remembered sets can also play cards. In Moss et al. [1993].

[Hosking *et al.*, 1992] Antony L. Hosking, J. Eliot B. Moss, and Darko Stefanović. A comparative performance evaluation of write barrier implementations. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 92–109, Vancouver, Canada, October 1992. *SIGPLAN Notices 27*, 10 (October 1992).

[Hudson and Moss, 1992] Richard L. Hudson and J. Eliot B. Moss. Incremental collection of mature objects. In Bekkers and Cohen [1992], pages 388–403.

[Hudson *et al.*, 1991] Richard L. Hudson, J. Eliot B. Moss, Amer Diwan, and Christopher F. Weight. A language-independent garbage collector toolkit. COINS Technical Report 91-47, University of Massachusetts, Amherst, September 1991.

[Jones and Lins, 1996] Richard Jones and Rafael Lins. *Garbage Collection: Algorithms for Automatic Dynamic Memory Management*. John Wiley, Chichester, 1996.

[Kolmogorov and Uspenskii, 1958] A. N. Kolmogorov and V. A. Uspenskii. On the definition of an algorithm. *Uspekhi Mat. Nauk*, 13:3–28, 1958. English translation in: *Russian Math. Surveys* **30** (1963) 217–245.

[Lang and Dupont, 1987] Bernard Lang and Francis Dupont. Incremental incrementally compacting garbage collection. In *SIGPLAN'87 Symposium on Interpreters and Interpretive Techniques*, volume 22(7) of *SIGPLAN Notices*, pages 253–263, Inst. Rech. Informat. Automat. Lab., France, 1987. ACM Press.

[LFP, 1990] *Conference Record of the 1990 ACM Symposium on Lisp and Functional Programming*, Nice, France, June 1990. ACM Press.

[LFP, 1994] *1994 ACM Conference on Lisp and Functional Programming*, Orlando, Florida, June 1994. ACM Press.

[Lieberman and Hewitt, 1983] Henry Lieberman and Carl Hewitt. A real-time garbage collector based on the lifetimes of objects. *Communications of the ACM*, 26(6):419–429, June 1983.

[McCarthy, 1960] John McCarthy. Recursive functions of symbolic expressions and their computation by machine, part I. *Communications of the ACM*, 3(4):184–195, April 1960.

[Minsky, 1963] Marvin L. Minsky. A Lisp garbage collector algorithm using serial secondary storage. Technical Report Memo 58 (rev.), Project MAC, MIT, Cambridge, MA, December 1963.

[Mohnen, 1995] Markus Mohnen. Efficient compile-time garbage collection for arbitrary data structures. Technical Report 95–08, University of Aachen, May 1995. Also in Seventh International Symposium on Programming Languages, Implementations, Logics and Programs, PLILP95.

[Moon, 1984] David Moon. Garbage collection in a large Lisp system. In *Proceedings of the ACM Symposium on Lisp and Functional Programming*, pages 235–246, Austin, TX, August 1984.

[Morrisett *et al.*, 1995] Greg Morrisett, Matthias Felleisen, and Robert Harper. Abstract models of memory management. In *Functional Programming and Computer Architecture*, June 1995.

[Moss *et al.*, 1993] Eliot Moss, Paul R. Wilson, and Benjamin Zorn, editors. *OOPSLA/ECOOP '93 Workshop on Garbage Collection in Object-Oriented Systems*, October 1993.

[Moss, 1987] J. Eliot B. Moss. Managing stack frames in Smalltalk. In *Proceedings of the ACM SIGPLAN '87 Symposium on Interpreters and Interpretive Techniques*, pages 229–240, St. Paul, Minnesota, July 1987. *SIGPLAN Notices 22*, 7 (July 1987).

[Nystrom, 1998] Nathaniel Nystrom. Bytecode-level analysis and optimization of Java class files. Master's thesis, Purdue University, West Lafayette, IN, May 1998.

[Odersky and Wadler, 1997] Martin Odersky and Philip Wadler. Pizza into Java: translating theory into practice. In *POPL '97. Proceedings of the 24th ACM SIGPLAN-SIGACT symposium on Principles of programming*, pages 146–159, New York, 1997. ACM Press.

[Proebsting *et al.*, 1998] Todd A. Proebsting, Gregg Townsend, Patrick Bridges, John H. Hartman, Tim Newsham, and Scott A. Watterson. Toba: Java for applications, a way ahead of time (WAT) compiler. (at http://www.cs.arizona.edu/sumatra/toba/), 1998.

[Reinhold, 1993] Mark B. Reinhold. *Cache performance of garbage-collected programming languages*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA, September 1993.

[Röjemo and Runciman, 1996] Niklas Röjemo and Colin Runciman. Lag, drag, void, and use: heap profiling and space-efficient compilation revisited. In *Proceedings of SIGPLAN'96 Conference on Programming Languages Design and Implementation*, volume 31 of *SIGPLAN Notices*, pages 34–41. ACM Press, 1996.

[Runciman and Röjemo, 1995] Colin Runciman and Niklas Röjemo. Lag, drag and post-mortem heap profiling. In *Implementation of Functional Languages Workshop*, Båstad, Sweden, September 1995.

[Runciman and Wakeling, 1992] Colin Runciman and David Wakeling. Heap profiling of lazy functional programs. Technical Report 172, University of York, Dept. of Computer Science, Heslington, York Y01 5DD, United Kingdom, April 1992.

[Sansom and Peyton Jones, 1994] Patrick M. Sansom and Simon L. Peyton Jones. Time and space profiling for non-strict, higher-order functional languages. Technical Report Research Report FP-1994-10, Dept. of Computing Science, University of Glasgow, Glasgow, Scotland, 1994.

[Sansom, 1994] Patrick M. Sansom. *Execution Profiling for Non-strict Functional Languages*. PhD thesis, University of Glasgow, Glasgow, Scotland, April 1994.

[Schönhage, 1980] A. Schönhage. Storage modification machines. *SIAM Journal on Computing*, 9:490–508, 1980.

[Seligmann and Grarup, 1995] Jacob Seligmann and Steffen Grarup. Incremental mature garbage collection using the train algorithm. In O. Nierstras, editor, *Proceedings of 1995 European Conference on Object-Oriented Programming*, Lecture Notes in Computer Science, pages 235–252, University of Aarhus, August 1995. Springer-Verlag.

[Shaw, 1988] Robert A. Shaw. *Empirical Analysis of a LISP System.* PhD thesis, Stanford University, February 1988. Available as Technical Report CSL-TR-88-351.

[Sleator and Tarjan, 1983] Daniel Dominic Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. In *Proceedings of the ACM SIGACT Symposium on Theory*, pages 235–245, Boston, Massachusetts, April 1983.

[Sobalvarro, 1988] Patrick G. Sobalvarro. A lifetime-based garbage collector for LISP systems on general-purpose computers, 1988. B.S. Thesis, Dept. of EECS, Massachusetts Institute of Technology, Cambridge.

[Stefanović and Moss, 1994] Darko Stefanović and J. Eliot B. Moss. Characterisation of object behaviour in Standard ML of New Jersey. In LFP [1994].

[Stefanović *et al.*, 1998a] Darko Stefanović, J. Eliot B. Moss, and Kathryn S. McKinley. Oldest-first garbage collection. Computer Science Technical Report 98-81, University of Massachusetts, Amherst, April 1998.

[Stefanović *et al.*, 1998b] Darko Stefanović, J. Eliot B. Moss, and Kathryn S. McKinley. On models for object lifetimes. February 1998.

[Stefanović, 1993a] Darko Stefanović. The garbage collection toolkit as an experimentation tool. In Moss et al. [1993].

[Stefanović, 1993b] Darko Stefanović. Generational copying garbage collection for Standard ML: a quantitative study. Master's thesis, Department of Computer Science, University of Massachusetts, 1993.

[Sullivan and Chillarege, 1991] Mark Sullivan and Ram Chillarege. Software defects and their impact on system availability – a study of field failures in operating systems. In *Digest of the 21st International Symposium on Fault Tolerant Computing*, pages 2–9, June 1991.

[Ungar and Jackson, 1988] David M. Ungar and Frank Jackson. Tenuring policies for generation-based storage reclamation. *SIGPLAN Notices*, 23(11):1–17, 1988.

[Ungar and Jackson, 1992] David M. Ungar and Frank Jackson. An adaptive tenuring policy for generation scavengers. *ACM Transactions on Programming Languages and Systems*, 14(1):1–27, 1992.

[Ungar, 1984] David Ungar. Generation scavenging: A non-disruptive high performance storage reclamation algorithm. In *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments*, pages 157–167, Pittsburgh, Pennsylvania, April 1984. *SIGPLAN Notices 19*, 5 (May 1984).

[Ungar, 1986] David M. Ungar. *The Design and Evaluation of a High Performance Smalltalk System*. ACM distinguished dissertation 1986. MIT Press, 1986.

[van Emde Boas, 1990] Peter van Emde Boas. *Machine Models and Simulations*, volume A: Algorithms and Complexity. Elsevier and MIT Press, 1990.

[Wilson and Moher, 1989] Paul R. Wilson and Thomas G. Moher. Design of the opportunistic garbage collector. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, pages 23–35, New Orleans, Louisiana, October 1989. *SIGPLAN Notices 24*, 10 (October 1989).

[Wilson *et al.*, 1991] Paul R. Wilson, Michael S. Lam, and Thomas G. Moher. Effective "static-graph" reorganization to improve locality in garbage-collected systems. In *Proceedings of the ACM SIGPLAN '91 Conference on Programming Language Design and Implementation*, pages 177–191, Toronto, Canada, June 1991. *SIGPLAN Notices 26*, 6 (June 1991).

[Wilson *et al.*, 1995] Paul R. Wilson, Mark S. Johnstone, Michael Neely, and David Boles. Dynamic storage allocation: A survey and critical review. In Henry Baker, editor, *Proceedings of International Workshop on Memory Management*, Lecture Notes in Computer Science, pages 1–116, Kinross, Scotland, September 1995. Springer-Verlag.

[Wilson, 1989] Paul R. Wilson. A simple bucket-brigade advancement mechanism for generation-based garbage collection. *SIGPLAN Notices*, 24(5):38–46, May 1989.

[Wilson, 1990] Paul R. Wilson. Some issues and strategies in heap management and memory hierarchies. In Eric Jul and Niels-Christian Juul, editors, *OOPSLA/ECOOP '90 Workshop on Garbage Collection in Object-Oriented Systems*, October 1990. Also in SIGPLAN Notices 23(1):45–52, January 1991.

[Wilson, 1992] Paul R. Wilson. Uniprocessor garbage collection techniques. In Bekkers and Cohen [1992], pages 1–42.

[Yi and Harrison, 1992] Kwangkeun Yi and Williams Ludwell Harrison, III. Interprocedural data flow analysis for compile-time memory management. Technical Report 1244, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Urbana, Illinois, August 1992.

[Zorn, 1989] Benjamin G. Zorn. *Comparative Performance Evaluation of Garbage Collection Algorithms*. PhD thesis, University of California at Berkeley, December 1989. Available as Technical Report UCB/CSD 89/544.

[Zorn, 1990a] Benjamin Zorn. Barrier methods for garbage collection. Technical Report CU-CS-494-90, University of Colorado at Boulder, November 1990.

[Zorn, 1990b] Benjamin Zorn. Comparing mark-and-sweep and stop-and-copy garbage collection. In LFP [1990].

[Zorn, 1991] Benjamin G. Zorn. The effect of garbage collection on cache performance. Technical Report CU-CS-528-91, University of Colorado at Boulder, May 1991.