A note on data access patterns in Standard ML of New Jersey.

Darko Stefanović

OSL+SAA Memo 1994-02-V1.
Version 1. May 4, 1994.

A note on data access patterns in Standard ML of New Jersey*

Darko Stefanović
Department of Computer Science
University of Massachussets

May 5, 1994

The increasing gap between processor speed and main memory speed has given rise to considerable interest in memory system performance of programs. A recent study[2] of programs written in Standard ML under the New Jersey implementation[1] pointed out that in a system with high allocation and few updates the allocated memory tends to be read back soon after allocation. The authors provide circumstantial evidence from cache simulation studies to support this assertion. This kind of access pattern favours different cache organisations[2] from a read-compute-write pattern that one may expect in traditional language settings.

Since proper understanding of the access patterns is seen to be important, we undertook to measure it directly. Our apparatus consists of the Standard ML of New Jersey compiler, version 0.93 (with the Sparc code generator), and the UMass Garbage Collector Toolkit, version 0.2, coupled with the Shade/Spixtools[1] instruction tracing tools, sixth edition. The benchmark programs are the same as in several previous studies of Standard ML.[2, 3]

We classify the read accesses as follows: accesses by C code (ML occasionally jumps there to execute run-time system primitives); accesses to old data not in the nursery; accesses to data in the nursery, below the current allocation pointer.[2] The last category is the one of interest in this study. For cache sizing issues, what matters is exactly the amount of allocation between the point of allocation of one object and the subsequent read access, not the intervening time or number of instructions. Table 1 summarises the accesses traced.

In Figure 1, the age of the first read access is shown on a cumulative plot. The words which are allocated but never accessed (this includes most object header words) are not considered here. For Leroy, 90% the words are read before another 100 words are allocated; 99% before 23500, and 99.9% before 59000. The average age of a word at the time it is first read is 19452 words, but the median is as low as 10. Similar patterns are observed in other programs.

*This work is supported by National Science Foundation grant CCR-9211272. The author can be reached via Internet address stefanov@cs.umass.edu

[1]We would like to thank Sun Microsystems for letting us use these tools.

[2]No garbage collection occurs in these traces

Program	Leroy	ML	PIA	Yacc
Instructions traced	33 665 867	43 243 104	46 386 236	39 724 300
Objects allocated	1 539 338	1 056 597	1 694 801	1 127 952
Bytes allocated	29 991 568	29 912 000	29 988 888	29 886 440
Allocating writes	7 499 644	7 496 041	7 498 587	7 547 540
Updating writes to the nursery	4 489	86 411	8 397	109 936
Reads from the nursery	6 095 289	7 013 274	9 056 133	7 857 202
Distinct nursery words read	4 141 975	4 657 689	5 073 062	4 926 840
Nursery words not read	3 307 669	2 838 352	2 425 525	2 620 700
Non-nursery reads	280 461	1 819 979	2 458 088	731 763
C code writes	3 304	7 037	4 314	5 944
C code reads	3 878	8 182	5 052	6 884

Table 1: Access characteristics of benchmark programs

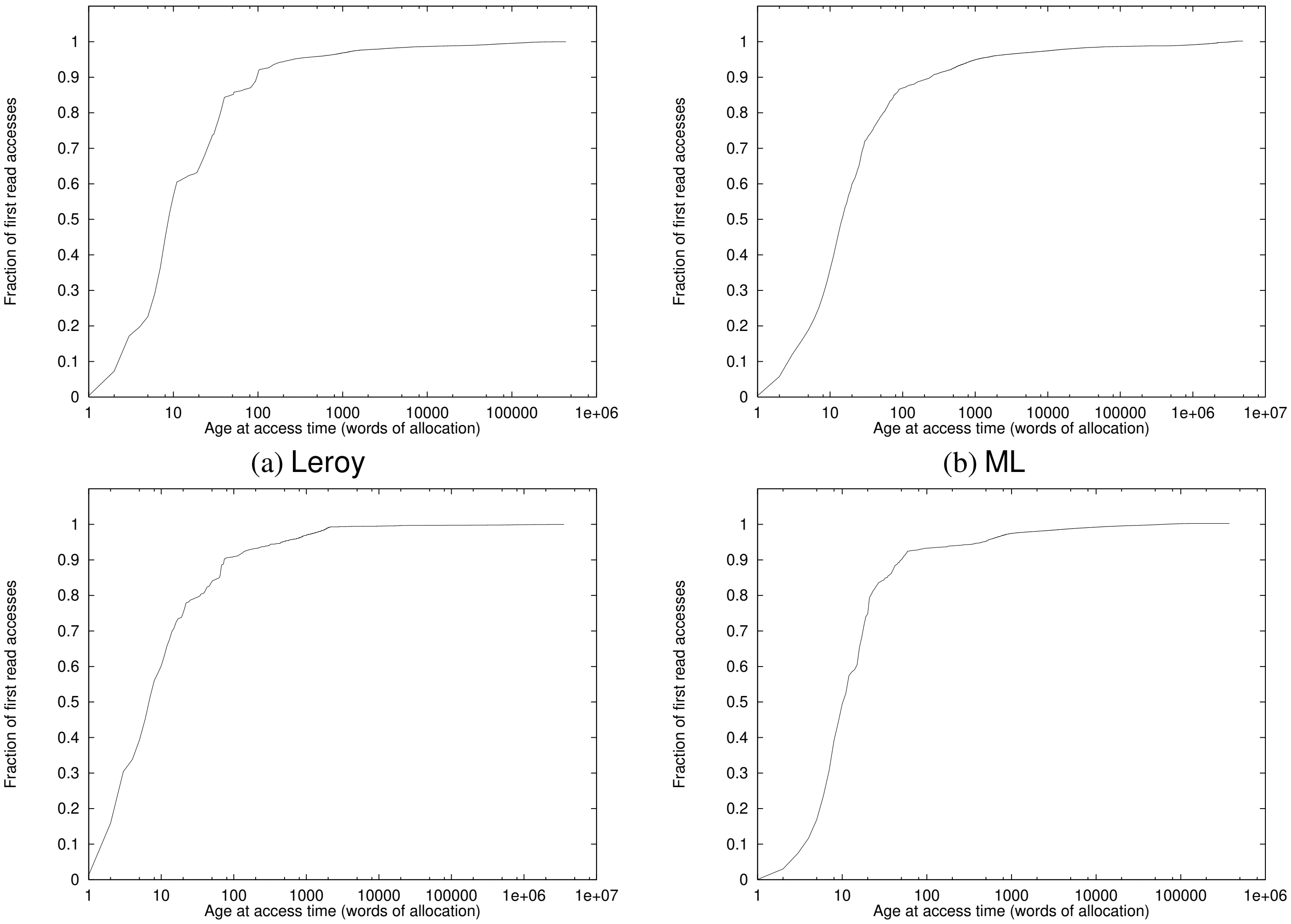
Thus, a 320K byte data cache, ignoring interference from accesses outside the nursery, will satisfy nursery first-read requests with a 0.1% miss rate. The effectiveness in satisfying reads beyond the first will be the subject of another note. An unfortunate consequence is that a small cache as found on some new workstations will necessarily have a high miss rate. For example, a modern 64-bit workstation with an 8K primary cache, which is 1000 words under a true 64-bit implementation of SML, would have a 3.5% read miss rate. The cost of read misses can be lowered with non-blocking caches if good instruction scheduling is possible; the short basic blocks of SML will not provide many opportunities for this, however. Therefore the read miss rate will be a significant performance-degrading factor.

References

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

[2] Amer Diwan, David Tarditi, and J. Eliot B. Moss. Memory subsystem performance of programs using copying garbage collection. In *Conference Record of the Twenty-First ACM Symposium on Principles of Programming Languages*, pages 1–14, Portland, Oregon, January 1994.

[3] Darko Stefanović and J. Eliot B. Moss. Characterisation of object behaviour in Standard ML of New Jersey. In *1994 ACM Conference on Lisp and Functional Programming*, Orlando, Florida, June 1994.



(a) Leroy (b) ML (c) PIA (d) Yacc

Figure 1: Time of first read access – cumulative distribution.