The International Symposium on Memory Management 2008
7-8 June, Tucson, Arizona

The ACM SIGPLAN International Symposium on Memory Management (ISMM) is a premier forum for research in the management of dynamically allocated memory. We interpret this remit widely: areas of interest include but are not limited to explicit storage allocation and deallocation; garbage collection algorithms and implementations; compiler analyses to aid memory management; interactions with languages, operating systems and hardware (especially the memory system); memory management related tools; and empirical studies of allocation and referencing behavior in programs that make significant use of dynamic memory.

Innovations in the Review Process

Although a small conference by comparison with PLDI or OOPSLA, ISMM has had a reputation ever since its inaugural meeting in Vancouver in October 1998 for a combination of high quality papers and a sympathetic and supportive community that makes it many people's favourite conference. ISMM'08 strove to ensure that its review process was as good as possible. We adopted five tactics.
(1) We appointed a strong Programme Committee (PC) with a wide range of interests and expertise.
(2) We used double-blind reviewing.
(3) We introduced a Review Committee (RC) to the process.
(4) We provided authors an opportunity for rebuttal.
(5) We adopted an "acceptance positive" approach.

In double-blind reviewing, the authors are anonymous to the reviewers, just as reviewers are anonymous to the authors. We used this process because research indicates that author anonymity may reduce bias [2,3] and lead to higher quality in accepted papers [1]. Authors were required to make reasonable efforts not to disclose their identities to reviewers, but for example to discuss their own prior work in the third person, as they would other related work. Authors were also allowed to provide reviewers with anonymous auxiliary material, such as proofs and source code, via the Programme Chair. Reviewers, for their part, were honour bound not to try to discover authors' identities, which were known only by the Programme Chair until a suitable point in the PC's deliberations.

To extend the depth and breadth of the reviewer pool beyond the Programme Committee, ISMM'08 introduced a "Review Committee". The RC is a more structured replacement for ad hoc expert reviews employed by a number of SIGPLAN conferences, including PLDI and ASPLOS. The committee was established ahead of time, by invitation from the Programme Chair. RC members differ from PC members in that they do not attend the PC meeting, they review a small number of papers, and they are not restricted in submitting papers to the conference. At ISMM'08, each paper received three PC reviews and one RC review, providing the authors with plenty of feedback while only requiring 3-way consensus at the PC meeting. The RC has a number of advantages over ad hoc external review assignment. (1) Each RC member reviewed multiple papers, so had some calibration. (2) Authors stated conflicts ahead of time and RC members bid for papers, making review assignments straightforward even with double-blind reviewing. (3) As a formally established committee, there was greater transparency and accountability. (4) The Programme Chair was able to draw systematically on past PC members, providing continuity and a great deal of perspective. (5) The reviewers received formal credit for their work. (6) Finally, submitting authors knew ahead of time who might review their paper (if an RC is impressive, this may help increase authors' confidence in the process). A further opportunity of the RC mechanism is that it could be used to review all PC paper submissions. We did not explore this at ISMM'08, but it may be useful, particularly for conferences such as PLDI where concerns about the potential for nepotism currently mean PC members may not submit to the conference at all.

Authors were provided an opportunity to respond succinctly to factual errors in reviews shortly before the Programme Committee met to make its decisions. Each PC member was asked to read each author response carefully ahead of time, and the PC was given 30 minutes at the start of its meeting to re-read them all. The discussion leader for each paper was asked to comment briefly on the response during the discussion so that the mechanism was not merely a token, but had a direct role in the decision making process. The responses had an impact on the outcome of a number of papers, and we believe most PC members found the responses useful.

Consensus-driven PC meetings can lead to conservative decisions, often quashing innovative work. In an effort to combat that problem, we followed a policy introduced by David Bacon at OOPSLA'07 of actively maintaining an "acceptance positive" stance within the PC. We therefore made a deliberate policy of asking the PC to try to find a way to accept those papers that were deemed especially interesting or innovative, even if the paper was seen to be flawed in other ways. To avoid compromising the traditionally high standards at ISMM, this meant the PC had to agree to shepherding any such paper to address its flaws. The result was that 5 of the 16 accepted papers were shepherded. Although this imposed a significant extra burden on the PC, we are convinced that it produced a stronger and more interesting programme.

A survey during the conference showed that ISMM'08 participants strongly supported these innovations and we commend them to the community. We also thank members of the PC and the RC for their hard work.

Organisation

Up until 2006, ISMM had met every two years, collocated with another major related conference such as OOPSLA or PLDI. Since 2006, ISMM has met every year. ISMM'08 was held in Tucson, Arizona, collocated with PLDI and other conferences and workshops. Although this meant that the interval between ISMM'07 (collocated with OOPSLA) was undesirably short (just 8 months), we considered it important to ensure that ISMM meet at a time each year which is both fixed and allows venues outside North America. Meeting in mid-summer allows ISMM to collocate with either PLDI or ECOOP, thus addressing both of these goals. Next year, ISMM will be held in conjunction with PLDI in Dublin, Ireland.

Although ISMM attendance dipped in 2007, it showed a modest increase (4%) this year. This was particularly encouraging since other collocated conferences saw significant falls in attendance: maybe this was due to a perception of location in Tucson in a summer month, though we enjoyed it! Perhaps the most significant statistic is that ISMM's attendance returned to its previous level of 20% of PLDI's (from a low of 14% in 2007).

Cooperation amongst collocated or related conferences is essential. As General and Programme Chairs of ISMM'08, we were grateful for the support of Rajiv Gupta, the PLDI General Chair. The most difficult issue was coordination of deadlines, which left and continues to leave something to be desired. ISMM's remit overlaps with that of PLDI, but we see it as a much more focussed conference. It is in the interest of both ISMM and PLDI that we do not compete for papers, but rather ensure that there is a timely venue for the very best papers. Because of the lengthy timescale between the published dates for submissions to PLDI'08 and author notification, we were forced to adopt a process that allowed authors who had already submitted a paper to PLDI'08 to submit the abstract to ISMM as well. However, any paper that was accepted at PLDI was automatically withdrawn from ISMM. Although this satisfies the SIGPLAN publication policy on dual submissions, it is far from ideal and places considerable pressure on the programme chair.

Collocation of focussed symposia such as ISMM and LCTES alongside our leading conferences such as PLDI is a positive model for SIGPLAN, with all parties benefiting from increased attendance, increased sharing and reduced travel. However, it requires a co-operative and co-ordinated approach that may require some flexibility from all parties. It seems that SIGPLAN, through its Executive, has an important and constructive role to play as a broker, ensuring the health of the entire SIGPLAN community.

The programme

The ISMM'08 programme was diverse. As well as papers on garbage collection and explicit deallocation, authors presented work on automatic management of non-memory resources, studies of memory locality, heap compression, object demographics, leak detection and models for bounding memory resources. Prizes for the best presentations were awarded to Jennifer Sartor, University of Texas and Filip Pizlo, Purdue University. It is particularly pleasing that both these speakers are students. The abstracts of all the papers are below.

The programme featured an excellent keynote presentation by David Bacon, IBM T.J. Watson Research Center, where he discussed the hard road from research idea to product, using the Metronome hard real-time garbage collector as an example. We also had excellent "Wild and Crazy" and "Student Lightning Talk" sessions. The Wild and Crazy Session was a fun and informal forum for discussion of interesting ideas in which presenters were given just 4 minutes in which to present their idea and 1 minute to take questions. Prizes were awarded for the craziest idea (Mike Bond, University of Texas) and the idea most worthy of implementation (Emery Berger, University of Massachusetts). Laurence Hellyer, University of Kent, was awarded the prize for the best student lightning talk (8 minutes plus two minutes for questions).

Putting together ISMM'08 was a team effort and we had a great team. First, we would like to thank all authors who submitted work to ISMM. We particularly wish to thank PC and RC members who did an extraordinary job under an extremely tight schedule. We would like to thank Dan Grossman for generously hosting the PC meeting at the University of Washington, and Julie Svendsen for working so hard to ensure that it went so smoothly. We particularly thank our keynote speaker, David Bacon, for his contribution to this year's programme. Finally, we would like to thank our sponsor ACM SIGPLAN and our supporters, Microsoft Research and Intel.

Richard Jones, General Chair
Steve Blackburn, Programme Chair

References

[1] D. N. Laband and M. J. Piette. Citation analysis of blinded peer review. *The Journal of the American Medical Association (JAMA): The Second International Congress on Peer Review in Biomedical Publication*, 272(2):147–149, July 1994.

[2] D. Watson, A. C. Andersen, and J. Hjorth. Mysterious disappearance of female investigators. *Nature*, 436(7048):174, July 2005. doi: 10.1038/436174a

[3] C. Wenneras and A. Wold. Nepotism and sexism in peer-review. *Nature*, 387(6 631):341–343, May 1997. doi:10.1038/387341a0

Abstracts

The CLOSER: Automating Resource Management in Java, Isil Dillig, Thomas Dillig, Eran Yahav and Satish Chandra
While automatic garbage collection has relieved programmers from manual memory management in Java-like languages, managing resources remains a considerable burden and a source of performance problems. In this paper, we present a novel technique for automatic resource management based on static approximation of resource lifetimes. Our source-to-source transformation tool, Closer, automatically transforms program code to guarantee that resources are properly disposed and handles arbitrary resource usage patterns. Closer generates code for directly disposing any resource whose lifetime can be statically determined; when this is not possible, Closer inserts conditional disposal code based on interest-reference counts that identify when the resource can be safely disposed. The programmer is only required to identify which types should be treated as resources, and what method to invoke to dispose each such resource. We successfully applied Closer on a moderate-sized graphics application that requires complex reasoning for resource management.

Parallel Generational-Copying Garbage Collection with a Block-Structured Heap, Simon Marlow, Tim Harris, Roshan James and Simon Peyton Jones
We present a parallel generational-copying garbage collector implemented for the Glasgow Haskell Compiler. We use a block-structured memory allocator, which provides a natural granularity for dividing the work of GC between many threads, leading to a simple yet effective method for parallelising copying GC. The results are encouraging: we demonstrate wall-clock speedups of on average a factor of 2 in GC time on a commodity 4-core machine with no programmer intervention, compared to our best sequential GC.

Limits of Parallel Marking Garbage Collection, Fridtjof Siebert
More and more, parallel multicore systems will be used even in low-end devices such as embedded controllers that require realtime guarantees. When garbage collection is used in these systems, parallel or concurrent garbage collection brings important performance advantages. In the context of realtime systems, it has to be shown that a parallel garbage collector implementation not only performs well in most cases, but guarantees on its performance in the worst case are required. This paper analyses the difficulties a parallel mark-and-sweep garbage collector faces during a parallel mark phase. The performance of such a garbage collector degrades when only some of the available processors can perform scanning work in the mark phase. Whenever the grey set contains fewer elements than the number of available processors, some processors may be stalled waiting for new objects to be added to the grey set. This paper gives an upper bound for the number of stalls that may occur as a function of simple properties of the memory graph. This upper bound is then determined for the Java applications that are part of the SPECjvm98 benchmark suite and the theoretical worst-case scalability of a parallel mark phase is analysed. The presented approach is then applied to a Java virtual machine that has uniform mark steps, which at first results in poor worst-case scalability. A small change in the implementation is then proposed and analysed to achieve good scalability even in the worst case.

Efficient Dynamic Heap Allocation of Scratch-Pad Memory, Ross McIlroy, Peter Dickman and Joe Sventek
An increasing number of processor architectures support scratch-pad memory - software managed on-chip memory. Scratch-pad memory provides low latency data storage, like on-chip caches, but under explicit software control. The simple design and predictable nature of scratchpad memories has seen them incorporated into a number of embedded and real-time system processors. They are also employed by multi-core architectures to isolate processor core local data and act as low latency inter-core shared memory. Managing scratch-pad memory by hand is time consuming, error prone and potentially wasteful; tools that automatically manage this memory are essential for its use by general purpose software. While there has been promising work in compile time allocation of scratch-pad memory, there will always be applications which require run-time allocation. Modern dynamic memory management techniques are too heavy-weight for scratch-pad management. This paper presents the Scratch-Pad Memory Allocator, a light-weight memory management algorithm, specifically designed to manage small on-chip memories. This algorithm uses a variety of techniques to reduce its memory footprint while still remaining effective, including: representing memory both as fixed-sized blocks and variable-sized regions within these blocks; coding of memory state in bitmap structures; and exploiting the layout of adjacent regions to dispense with boundary tags for split and coalesce operations. We compare the performance of this allocator against Doug Lea's malloc implementation for the management of core-local and inter-core shared scratchpad memories under real world memory traces. This algorithm manages small memories efficiently and scales well under load when multiple competing cores access shared memory.

Supporting Superpage Allocation without Additional Hardware Support, Mel Gorman and Patrick Healy
Today, many modern processors support more than one page size. The larger pages, called superpages, have been identified as one means of reducing the time spent servicing translation lookaside buffer (TLB) misses in the early 1990s by increasing TLB reach. Widespread usage of superpages has been limited by the requirement that superpages consist of physically contiguous and naturally-aligned small pages. This makes external fragmentation a serious problem for an operating system, one that is almost non-existent when processes use only one page size. Hardware solutions to mitigate this limitation such as sub-blocking, shadow page-tables and a variety of hybrid solutions have not seen wide-spread adoption. This has curtailed automatic superpage support as it is known that superpage availability will decrease during the system's lifetime as external fragmentation grows. This paper presents a placement policy for an operating system's physical page allocator to mitigate

external fragmentation problems by grouping pages based on the system's ability to relocate the data. Secondly, the necessary changes to the page reclamation algorithm for it to be contiguity-aware are described while minimising impact to the reclamation algorithms' normal decisions. The performance impact on different machine types is illustrated and it is shown that the superpage allocation success rate is improved. These mechanisms are complementary to any of the hardware solutions proposed in the past.

Memory Management for Self-Adjusting Computation, Matthew Hammer and Umut Acar
The cost of reclaiming space with traversal-based garbage collection is inversely proportional to the amount of free memory, i.e., O(1/(1-f)), where f is the fraction of memory that is live. Consequently, the cost of garbage collection can be very high when the size of the live data remains large relative to the available free space. Intuitively, this is because allocating a small amount of memory space will require the garbage collector to traverse a significant fraction of the memory only to discover little garbage. This is unfortunate because in some application domains the size of the memory-resident data can be generally high. This can cause high GC overheads, especially when generational assumptions do not hold. One such application domain is self-adjusting computation, where computations use memory-resident execution traces in order to respond to changes to their state (e.g., inputs) efficiently. This paper proposes memory-management techniques for self-adjusting computation that remain efficient even when the size of the live data is large. More precisely, the proposed techniques guarantee O(1) amortized cost for each reclaimed memory object. We propose a set of primitives for self-adjusting computation that support the proposed memory management techniques. The primitives provide an operation for allocating memory; we reclaim unused memory automatically. We implement a library for supporting the primitives in the C language and perform an experimental evaluation. Our experiments show that the approach can be implemented with reasonably small constant-factor overheads and that the programs written using the library behave optimally. Compared to previous implementations, we measure up to an order of magnitude improvement in performance and up to a 75% reduction in space usage.

Runtime Support for Region-Based Memory Management in Mercury, Quan Phan, Gerda Janssens and Zoltan Somogyi
Applying region-based memory management (RBMM) to logic programming languages poses a special challenge: backtracking can require regions removed during forward execution to be ``resurrected'', and any memory allocated during a computation that has been backtracked over must be recovered promptly, without waiting for the regions involved to come to the end of their life. In this paper, we describe how we implemented runtime support for RBMM in the logic programming language Mercury, whose specialized implementation of the language constructs involved in backtracking required equally specialized support. Our benchmark Mercury programs run about 25% faster on average with RBMM than with the usual Boehm garbage collector, and for some programs, RBMM achieves optimal memory consumption.

A Reference Counting Garbage Collection Algorithm for Cyclical Functional Programming, Baltasar Trancon y Widemann
Reference-counting garbage collection is known to have problems with the collection of cyclically connected data. There are two historically significant styles of cycle-aware algorithms: The style of Brownbridge that maintains a subset of marked edges and the invariant that every cycle contains at least one marked edge, and the style of Martinez-Lins-Wachenchauzer (MLW) that involves local mark-and-scan procedures to detect cycles. The former is known to be difficult to design and implement correctly, and the latter to have pathological efficiency for a number of very typical situations. We present a novel algorithm that combines both approaches to obtain reasonably efficient local mark-and-scan phases with a marking invariant that is rather cheap to maintain. We demonstrate that the assumptions of this algorithm about mutator activity patterns make it well-suited, but not limited, to a functional programming technique for cyclic data. We evaluate the approach in comparison with simple and more sophisticated MLW algorithms using a simple benchmark based on that functional paradigm.

Path Specialization: Reducing Phased Execution Overheads, Filip Pizlo, Erez Petrank and Bjarne Steensgaard
As garbage collected languages become widely used, the quest for reducing collection overheads becomes essential. In this paper, we propose a compiler optimization called path specialization that shrinks the cost of memory barriers for a wide variety of garbage collectors including concurrent, incremental, and real-time collectors. Path specialization provides a non-trivial decrease in write-barrier overheads and a drastic reduction of read-barrier overheads. It is effective when used with collectors that go through various phases each employing a different barrier behavior, and is most effective for collectors that have an idle phase, in which no barrier activity is required. We have implemented path specialization in the Bartok compiler and runtime for C# and tested it with state-of-the-art concurrent and real-time collectors, demonstrating its efficacy.

Sampling-based Program Locality Approximation, Yutao Zhong and Wentao Chang Reuse signature, or reuse distance pattern, is an accurate model for program memory accessing behaviors. It has been studied and shown to be effective in program analysis and optimizations by many recent works. However, the high overhead associated with reuse distance measurement restricts the scope of its application. This paper explores applying sampling in reuse signature collection to reduce the time overhead. We compare different sampling strategies and show that an enhanced systematic sampling with a uniform coverage of all distance ranges can be used to extrapolate the reuse distance distribution. Based on that analysis, we present a novel sampling method with a measurement accuracy of more than 99%. Our average speedup of reuse signature

collection is 7.5 while the best improvement observed is 34. This is the first attempt to utilize sampling in measuring reuse signatures. Experiments with varied programs and instrumentation tools show that sampling has great potential in promoting the practical uses of reuse signatures and enabling more optimization opportunities.

Memory Pooling Assisted Data Splitting (MPADS), Stephen Curial, Peng Zhao, Jose Nelson Amaral, Yaoqing Gao, Shimin Cui, Raul Silvera and Roch Archambault
This paper describes Memory-Pooling-Assisted Data Splitting (MPADS), a framework that combines data structure splitting with memory pooling. Although MPADS may call to mind memory padding, a distinction of this framework is that is does not insert padding. MPADS relies on pointer analysis to ensure that splitting is safe and applicable to type-unsafe language. MPADS makes no assumption about type safety. The analysis can identify cases in which the transformation could lead to incorrect code and thus MPADS abandons those cases. To make data structure splitting efficient in a commercial compiler, MPADS is designed with great attention to reduce the number of instructions required to access the data after the data-structure splitting. Moreover the implementation of MPADS reveals that architecture details should be considered carefully when re-arranging data allocation. For instance one of the most significant gains from the introduction of data-structure splitting in code targetting the IBM POWER architecture is a dramatic decrease in the amount of data prefetched by the hardware prefetch engine without a noticeable decrease in the cache utilization. Triggering fewer hardware prefetch streams frees memory bandwidth and cache space. Fewer prefetching streams also reduce the interference between the data accessed by multiple cores in modern multicore processors.

No Bit Left Behind: Limits of Heap Data Compression, Jennifer B. Sartor, Martin Hirzel and Kathryn S. McKinley
On one hand, the high cost of memory continues to drive demand for memory efficiency on embedded and general purpose computers. On the other hand, programmers are increasingly turning to managed languages like Java for their functionality, programmability, and reliability. Managed languages, however, are not known for their memory efficiency, creating a tension between productivity and performance. This paper examines the sources and types of memory inefficiencies in a set of Java benchmarks. Although prior work has proposed specific heap data compression techniques, they are typically restricted to one model of inefficiency. This paper generalizes and quantitatively compares previously proposed memory-saving approaches and idealized heap compaction. It evaluates a variety of models based on strict and deep object equality, field value equality, removing bytes that are zero, and compressing fields and arrays with a limited number and range of values. The results show that substantial memory reductions are possible in the Java heap. For example, removing bytes that are zero from arrays is particularly effective, reducing the application's memory footprint by 44% on average. We are the first to combine multiple savings models on the heap, which very effectively reduces the application by up to 86%, on average 62%. These results demonstrate that future work should be able to combine a high productivity programming language with memory efficiency.

A Study of Java Object Demographics, Richard Jones and Chris Ryder
Researchers have long strived to exploit program behaviour in order to improve garbage collection efficiency. For example, by using a simple heuristic, generational GC manages short-lived objects well, although longer-lived objects will still be promoted to an older generation and may be processed repeatedly thereafter. In this paper, we provide a detailed study of Java object lifetimes which reveals a richer landscape than the generational view offers. Allocation site has been claimed to be a good predictor for object lifetime, but we show that object lifetime can be categorised more precisely than 'short-lived/long-lived/immortal'. We show that (i) sites allocate objects with lifetimes in only a small number of narrow ranges, and (ii) sites cluster strongly with respect to the lifetime distributions of the objects they allocate. Furthermore, (iii) these clusterings are robust against the size of the input given to the program and (iv) are likely to allocate objects that are live only in particular phases of the program's execution. Finally, we show that, in contrast to previous studies, (v) allocation site alone is not always sufficient as a predictor of object lifetime distribution but one further level of stack context suffices.

Practical Memory Leak Detector Based on Parameterized Procedural Summaries, Yungbum Jung and Kwangkeun Yi
We present a static analyzer that detects memory leaks in C programs. It achieves relatively high accuracy at a relatively low cost on SPEC2000 benchmarks and several open-source software packages, demonstrating its practicality and competitive edge against other reported analyzers: for a set of benchmarks totalling 1,777 KLOCs, it found 332 bugs with 47 additional false positives (a 12.4% false-positive ratio), and the average analysis speed was 720 LOC/sec. We separately analyze each procedure's memory behavior into a summary that is used in analyzing its call sites. Each procedural summary is parameterized by the procedure's call context so that it can be instantiated at different call sites. What information to capture in each procedural summary has been carefully tuned so that the summary should not lose any common memory-leak-related behaviors in real-world C programs. Because each procedure is summarized by conventional fixpoint iteration over the abstract semantics (a la abstract interpretation), the analyzer naturally handles arbitrary call cycles from direct or indirect recursive calls.

*Parametric Prediction of Heap Memory Requirement*s, Víctor Braberman, Federico Fernández, Diego Garbervetsky and Sergio Yovine
This work presents a technique to compute symbolic polynomial approximations of the amount of dynamic memory required to safely execute a method without running out of memory, for Java-like imperative programs. We consider object allocations and deallocations made by the method and the methods it transitively calls. More precisely, given an initial

configuration of the stack and the heap, the peak memory consumption is the maximum space occupied by newly created objects in all states along a run from it. We over-approximate the peak memory consumption using a scoped-memory management where objects are organized in regions associated with the lifetime of methods. We model the problem of computing the maximum memory occupied by any region configuration as a parametric polynomial optimization problem over a polyhedral domain and resort to Bernstein basis to solve it. We apply the developed tool to several benchmarks.

Analysing Memory Resource Bounds for Bytecode Programs, Wei-Ngan Chin, Huu Hai Nguyen, Corneliu Popeea and Shengchao Qin

Embedded systems are becoming more widely used but these systems are often resource constrained. Programming models for these systems should take into formal consideration resources such as stack and heap. In this paper, we show how memory resource bounds can be inferred for assembly-level programs. Our inference process captures the memory needs of each method in terms of the symbolic values of its parameters. For better precision, we infer path-sensitive information through a novel guarded expression format. Our current proposal relies on a Presburger solver to capture memory requirements symbolically, and to perform fixpoint analysis for loops and recursion. Apart from safety in memory adequacy, our proposal can provide estimate on memory costs for embedded devices and improve performance via fewer runtime checks against memory bound.