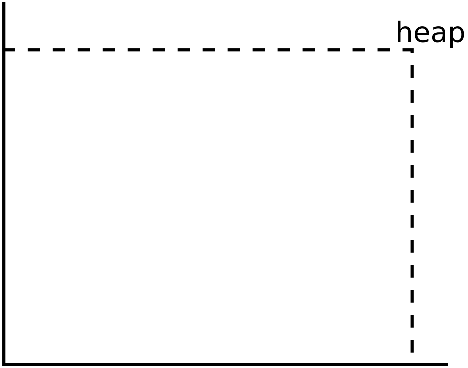
Short-term Memory for Self-collecting Mutators

Martin Aigner, **Andreas Haas**, Christoph M. Kirsch,
Michael Lippautz, Ana Sokolova,
Stephanie Stroka, Andreas Unterweger

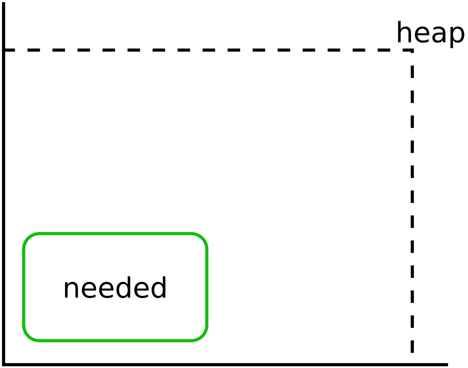
University of Salzburg

October 12, 2010

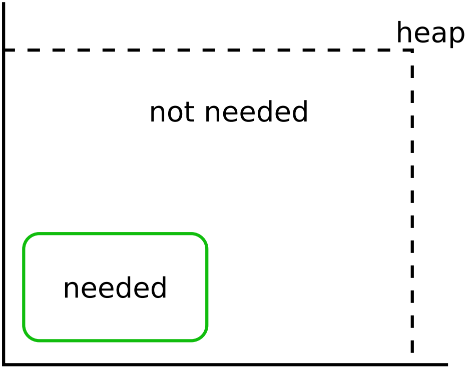
Heap Management



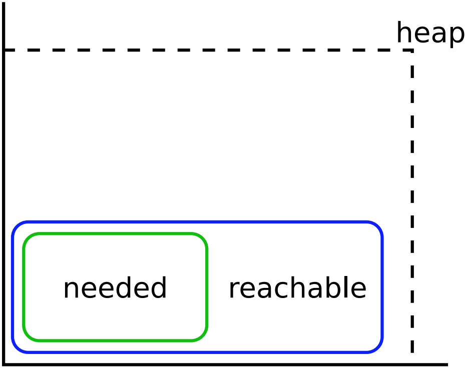
Heap Management



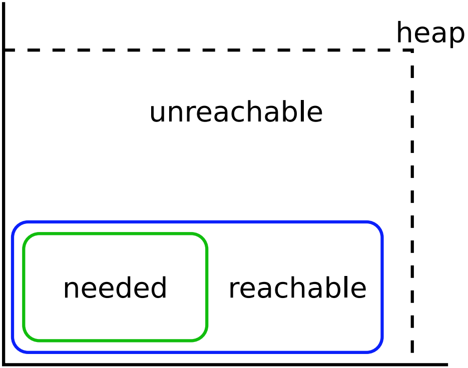
Heap Management



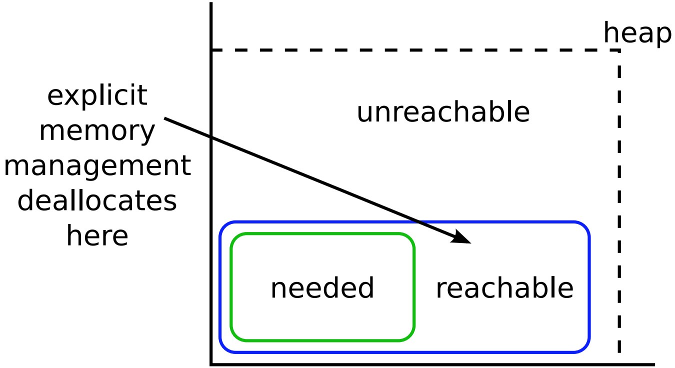
Heap Management



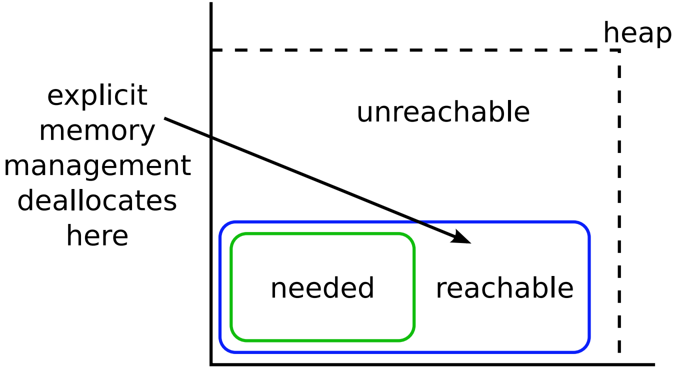
Heap Management



Heap Management

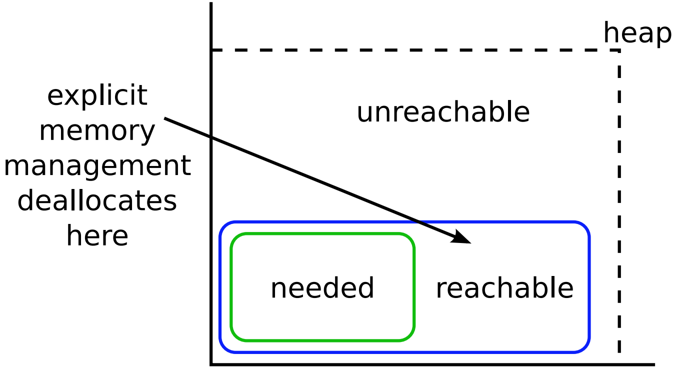


Heap Management



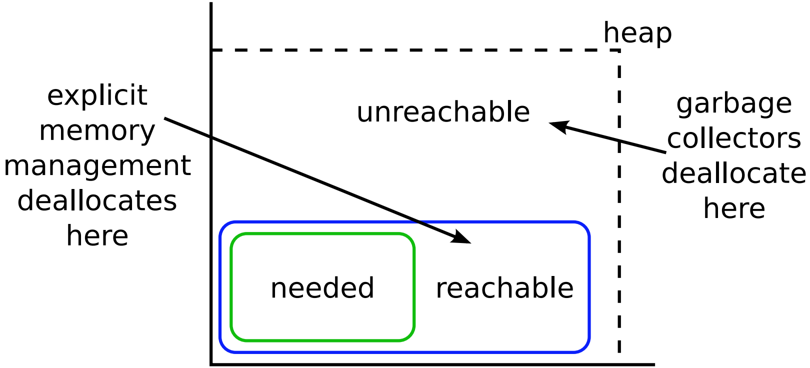
- memory leaks

Heap Management



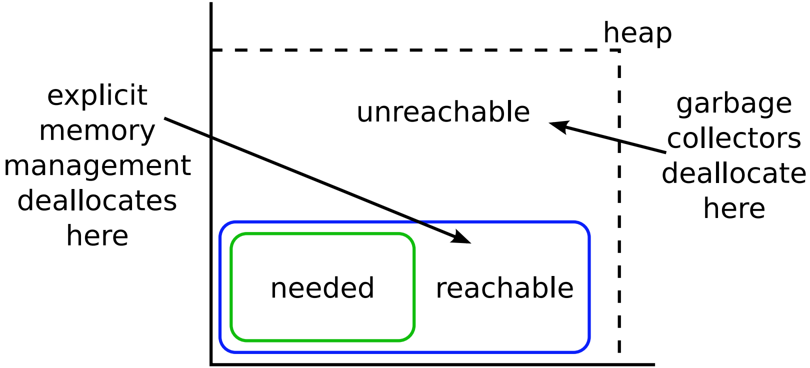
- memory leaks
- dangling pointers

Heap Management



- memory leaks
- dangling pointers

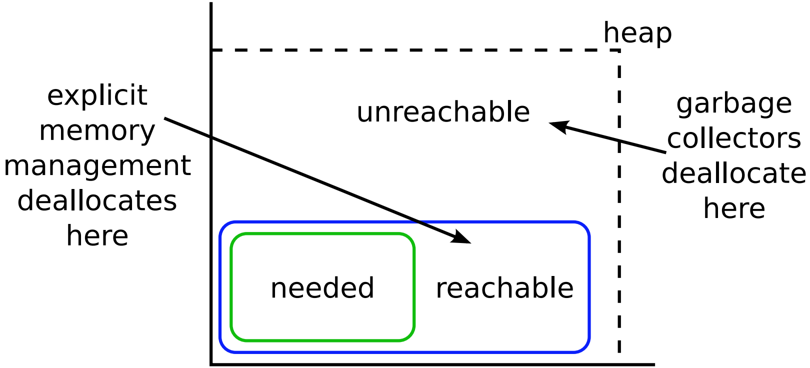
Heap Management



- memory leaks
- dangling pointers

- tracing

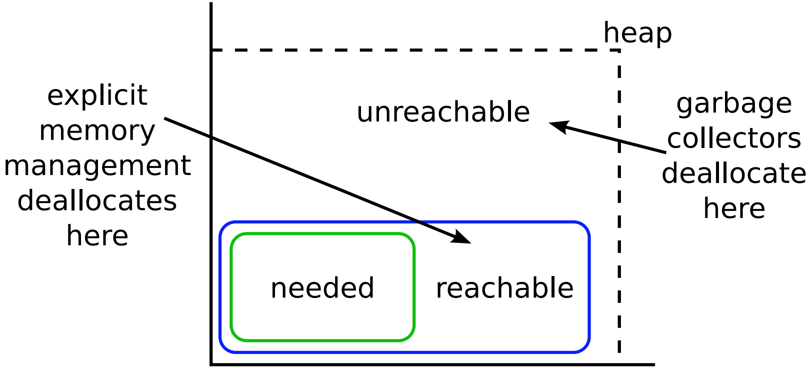
Heap Management



- memory leaks
- dangling pointers

- tracing
- reference counting

Heap Management



- memory leaks
- dangling pointers

- tracing
- reference counting
- reachable memory leaks

Persistent Memory Model

- Allocated memory objects are guaranteed to exist until deallocation
- Explicit deallocation is not safe (dangling pointers) and can be space-unbounded (memory leaks)
- Implicit deallocation (unreachable objects) is safe but may be slow or space-consuming (proportional to the size of live memory) and can still be space-unbounded (memory leaks)

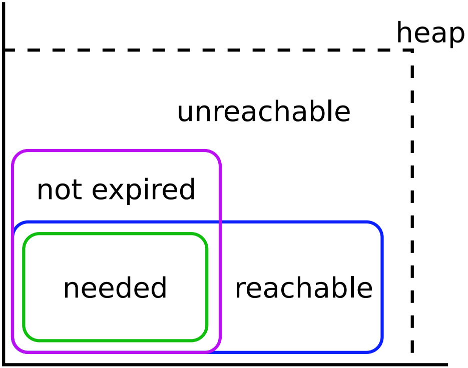
Short-term Memory

- Memory objects are only guaranteed to exist for a finite amount of time
- Memory objects are allocated with a given expiration date
- Memory objects are neither explicitly nor implicitly deallocated but may be refreshed to extend their expiration date

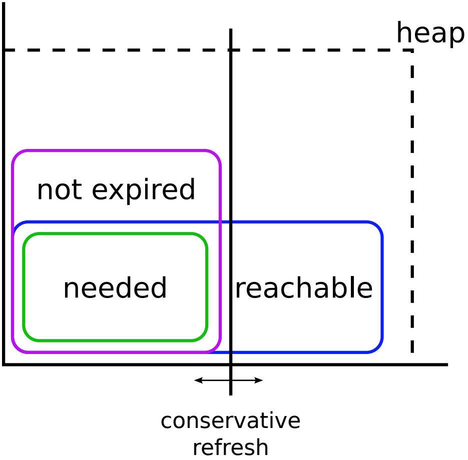
Short-term Memory

With short-term memory programmers or algorithms
specify which memory objects are still needed
and not
which memory objects are
not needed anymore!

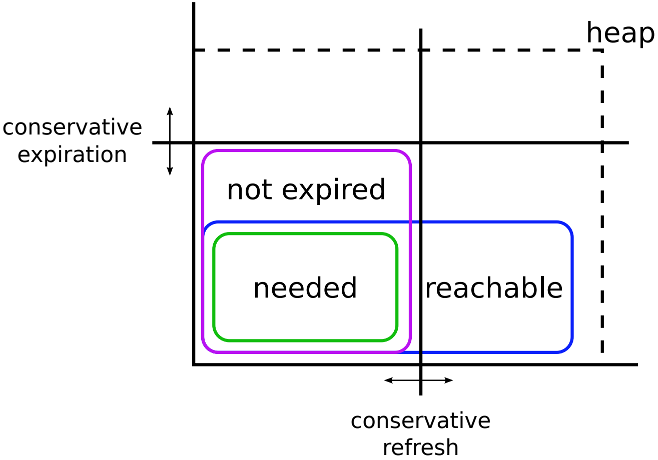
Short-term Memory



Short-term Memory



Short-term Memory



Short-term Memory - Sources of Errors

- Memory leaks
  - When not-needed objects are continuously refreshed
  - When time does not advance
- Dangling Pointers
  - When needed objects are not refreshed

Self-collecting Mutators

Programming Model

- Explicit memory management
  - The programmer (or an algorithm) adds memory management calls to the program code
- Hybrid approach for backward compatibility
  - Per default objects are allocated as persistent and managed by the existing memory management (malloc/free, garbage collection)
  - A refresh-call makes an object short-term, e.g. the objects gets an expiration date

Programming Model - Expiration Date

- An object gets an expiration date when it gets refreshed and is then managed by our system
- A programmer refreshes objects explicitly
  - Every refresh-call creates a new expiration date for an object
    - The object expires when all its expiration dates are expired

Programming Model - Time

- A software clock is used for object expiration
  - An integer counter which is increased by tick-calls
  - An expiration date has expired when its value is less than the time of the software clock
- Every thread has its own thread-local clock
  - Expiration dates expire according to the clock of the thread which created the expiration date

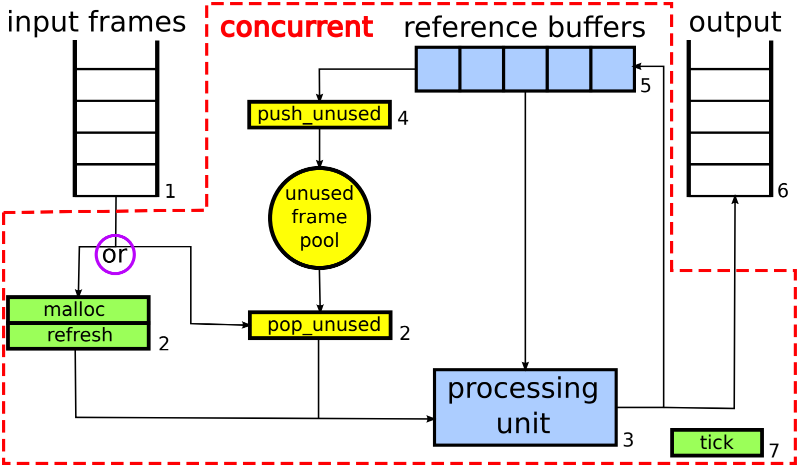
Example - Monte Carlo

```
monteCarlo(int repetitions)
{
    Vector results = new Vector(repetitions);
    for(int i = 0; i < repetitions; i++)
    {
        RandomWalk walk = createRandomWalk();

        results.add(doCalculation(walk);

    }

    evaluateResults(results);
}
```

Example - Monte Carlo

```
monteCarlo(int repetitions)
{
    Vector results = new Vector(repetitions);
    for(int i = 0; i < repetitions; i++)
    {
        RandomWalk walk = createRandomWalk();
        SCM.refresh(walk, 0);
        results.add(doCalculation(walk));
        SCM.tick();
    }

    evaluateResults(results);
}
```

Example - x264 Video Encoder



Other Use Cases

benchmark	LoC	tick	refresh	free	aux	total
mpg123	16043	1	0	(-)43	0	44
JLayer	8247	1	6	0	2	9
Monte Carlo	1450	1	3	0	2	6
LuIndex	74584	2	15	0	3	20

Table: Use cases of short-term memory: lines of code of the benchmark, number of tick-calls, number of refresh-calls, number of free-calls, number of auxiliary lines of code, and total number of modified lines of code.

Implementation

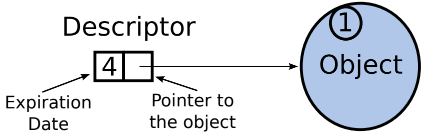
Implementation

- Our implementation is called self-collecting mutators (SCM)
  - The threads (mutators) of a program collect their expired objects by themselves
  - At memory management calls a constant number of expired objects are collected
- We have implementations in C, Java and Go
- The C implementation is based on ptmalloc2
- The Java implementation is based on the Jikes RVM
- For the Go implementation we extended the 6g Go runtime

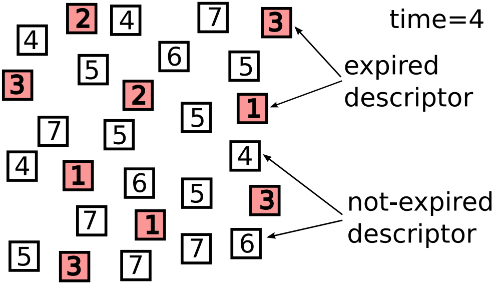
Available at: tiptoe.cs.uni-salzburg.at/short-term-memory

Implementation - Descriptors

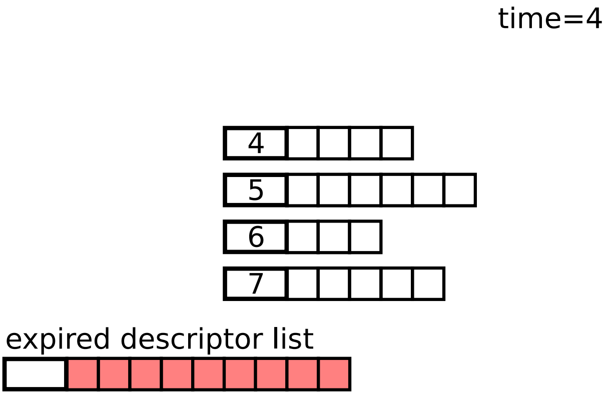
- An Object can have multiple expiration dates
  - An expiration date is represented by a descriptor, which stores the expiration date and a pointer to the object
- For each expiration date of an object there exists one descriptor
  - Every object contains a descriptor counter (1 word) in its header which counts the number of descriptors pointing to it



Implementation - Descriptors Buffer

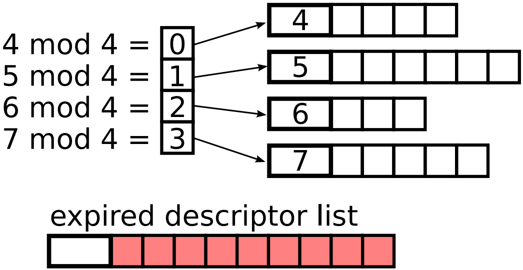


Implementation - Descriptor Buffer

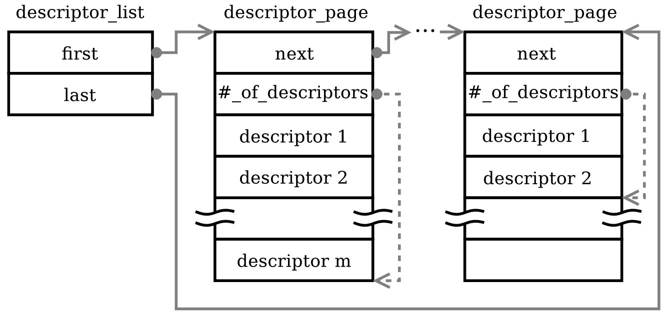


Implementation - Descriptor Buffer

time = 4
maximal expiration extension = 3



Implementation - Descriptor Buffer



Implementation - malloc/new

Definition

malloc(size)
new Object()

- The malloc/new call of SCM increases the requested amount of memory by one word and uses the underlying memory management then to allocate the memory
  - In C it is ptmalloc2
  - In Java and Go it is a mark-sweep garbage collector
- The additional header word of an object is used for the descriptor counter
- The descriptor counter is initialized with zero

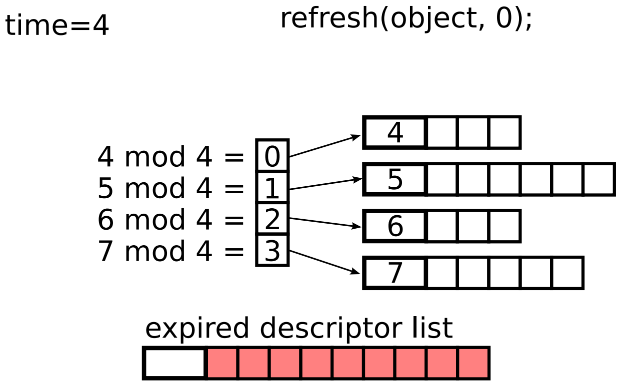
Implementation - refresh

Definition

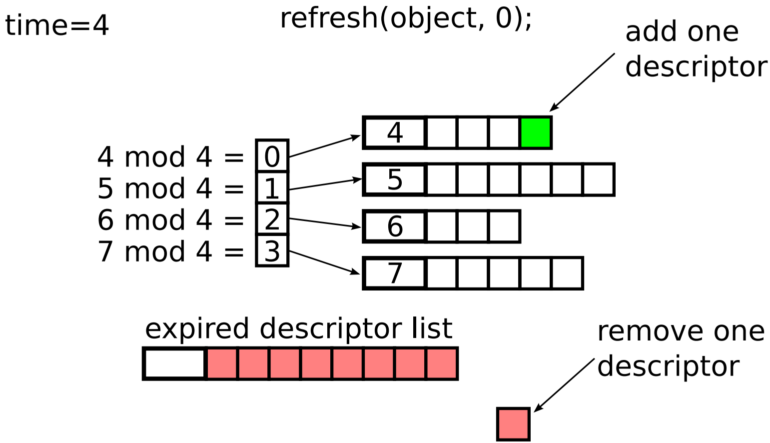
refresh(object, extension)

- The refresh-call creates a new descriptor for the considered object
- The refresh-call consists of four operations:
  1. Increase the descriptor counter of the specified object
  2. Create a new descriptor and store it in the descriptor buffer which corresponds to the given expiration extension.
  3. (Self-collection) Remove one descriptor from the expired_descriptors_list
  4. Decrease the descriptor counter of the object which corresponds to that descriptor
     - If the descriptor counter gets zero, the object is deallocated

Before refresh



After refresh



Implementation - tick

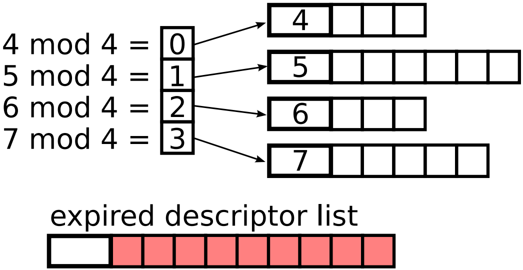
Definition

tick()

- The tick-call increases the thread-local time
- The descriptor list which expires by that time advance is appended to the expired descriptor list

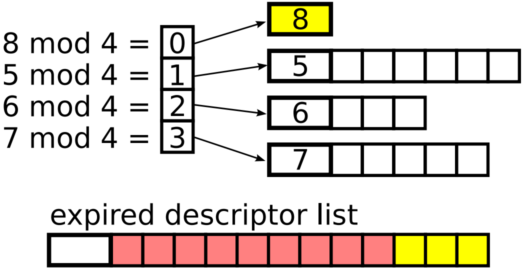
Before tick

time=4



After tick

time=5



Implementation - free

Definition

free(object)

- In C it is still possible to use the free-call for explicit deallocation
- When the descriptor counter of the object is zero, the object is deallocated
- Otherwise nothing is done
    - The object will be deallocated when its last descriptor expires

Implementation - Garbage Collector

- All objects are considered to compute reachability
- Short-term objects are not deallocated
    - They will be deallocated when their last descriptor expires

Experiments

What we want to show

- In C
  - Short-term memory is easier to use while not loosing temporal performance and with low memory overhead
- In Java and Go
  - Short-term memory is more difficult to use but improves temporal performance (e.g. reduce the number of garbage collection runs)

Experiments - System Configuration

CPU	2x AMD Opteron DualCore, 2.0 GHz
RAM	4GB
OS	Linux 2.6.32-21-generic
Java VM	Jikes RVM 3.1.0
C compiler	gcc version 4.4.3
C allocator	ptmalloc2-20011215 (glibc-2.10.1)

Table: System configuration.

Monte Carlo - Total Runtime

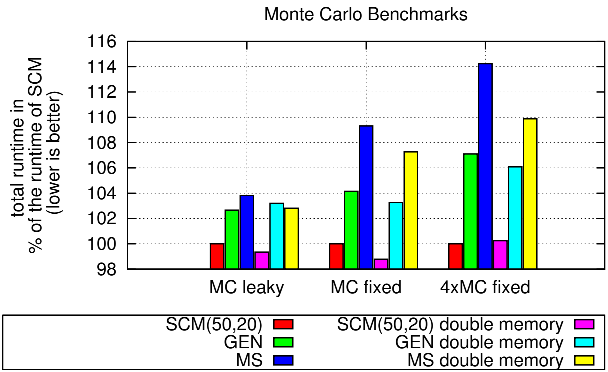


Figure: Total execution time of the Monte Carlo benchmarks in percentage of the total execution time of the benchmark using self-collecting mutators.

Monte Carlo - Latency and Free Memory

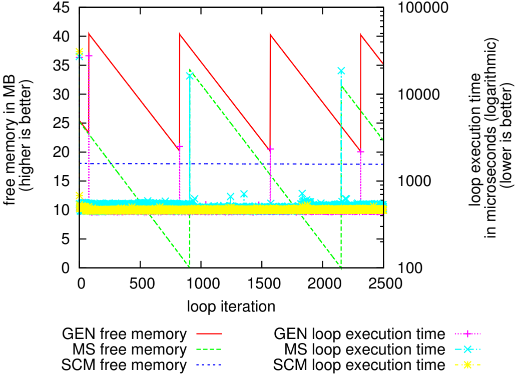


Figure: Free memory and loop execution time of the fixed Monte Carlo benchmark.

Monte Carlo - Tick Frequency - Latency

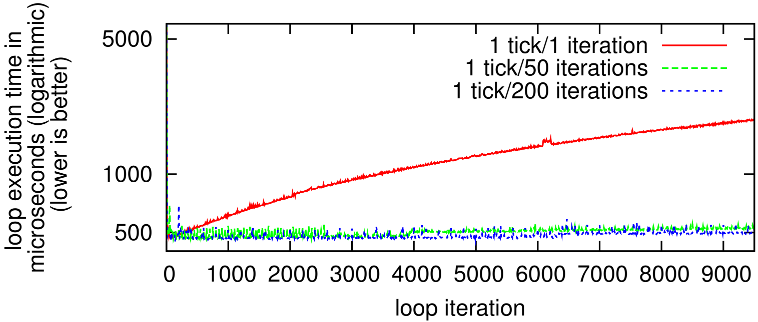


Figure: Loop execution time of the Monte Carlo benchmark with different tick frequencies. Self-collecting mutators is used.

Monte Carlo - Tick Frequency - Free Memory

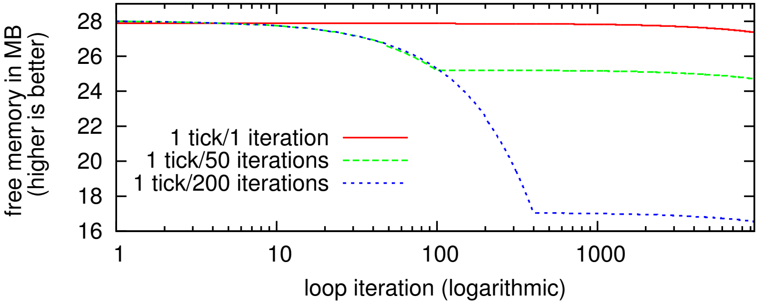


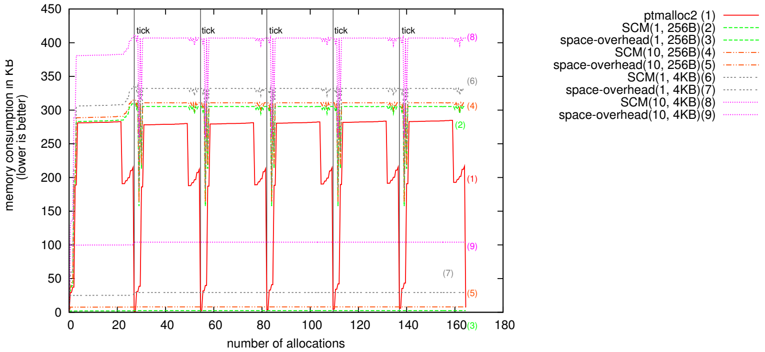
Figure: Free memory of the Monte Carlo benchmark with different tick frequencies. Self-collecting mutators is used.

mpg123 - Total Runtime

ptmalloc2	895.25ms	100.00%
ptmalloc2 through SCM	899.43ms	100.47%
SCM(1, 256B)	890.18ms	99.43%
SCM(10, 256B)	898.28ms	100.34%
SCM(1, 4KB)	892.18ms	99.66%
SCM(10, 4KB)	892.28ms	99.67%

Table: Total execution times of the mpg123 benchmark averaged over 100 repetitions. Here, SCM(n, m) stands for self-collecting mutators with a maximal expiration extension of n and descriptor page size m.

mpg123 - Memory Consumption



Memory overhead and consumption of the mpg123 benchmark. Again, SCM(n, m) stands for self-collecting mutators with a maximal expiration extension of n and descriptor page size m. We write space-overhead(n, m) to denote the memory overhead of the SCM(n, m) configurations for storing descriptors and descriptor counters.

Thank You

check out:

eurosys2011.cs.uni-salzburg.at