

Automatic Region-based Memory Management for Real-time Embedded Systems

Guillaume Salagnac

► **To cite this version:**

Guillaume Salagnac. Automatic Region-based Memory Management for Real-time Embedded Systems. Conference on Languages, Compilers, and Tools for Embedded Systems, 2006, Ottawa, Canada. 2006. <inria-00486735>

HAL Id: inria-00486735

<https://hal.inria.fr/inria-00486735>

Submitted on 26 May 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Automatic Region-Based Memory Management for Real-Time Embedded Systems



Guillaume Salagnac
Verimag - Grenoble - France

<http://www-verimag.imag.fr/PEOPLE/Guillaume.Salagnac/>

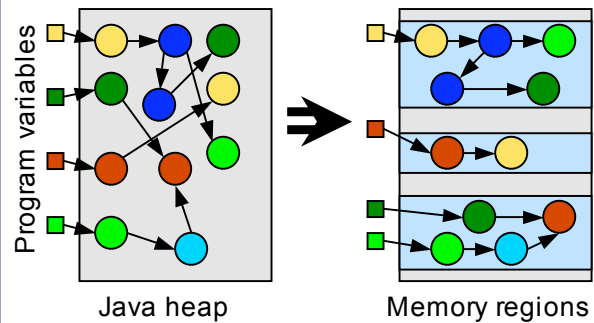
Motivation :

- Java means automatic memory management
- Garbage Collector means problems in a real-time context
 - Unpredictable pause times
 - Fragmentation of the heap
- *How can we provide automatic memory management without using a GC ?*

Our approach :

- Use region-based memory management
- Group data structures in regions
- Use a compile-time analysis to place objects in regions

Memory management with regions



- Objects allocated side by side : no more fragmentation
- Regions destroyed as a whole : predictable times
- Drawback: each object must be placed when allocated

Pointer Interference Analysis :

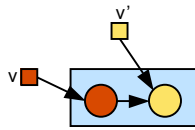
- Build a partition of local variables
- $v \sim v'$ means they belong to the same data structure
- Simple algorithm :

$$\begin{aligned} v &:= u && \implies v \sim u \\ v &:= u.f && \implies v \sim u \\ v.f &:= u && \\ m() & && \\ \text{call } \begin{cases} p1 \sim v1 \\ p2 \sim v2 \end{cases} &&& \implies v1 \sim v2 \\ m'() & && \text{if } p1 \sim p2 \end{aligned}$$

Allocation Policy :

Simple allocation policy :
if two variables verify $v \sim v'$,
place their objects in the same region

Data structures will be automatically grouped by region



Other kinds of pointer analysis :

Escape analysis :

Does my object live longer than its method of origin ?

Points-to analysis :

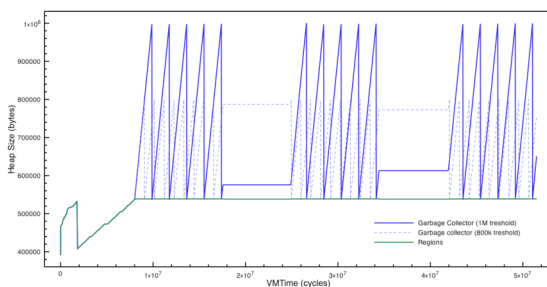
Where do the objects of my variable come from ?

Purity analysis :

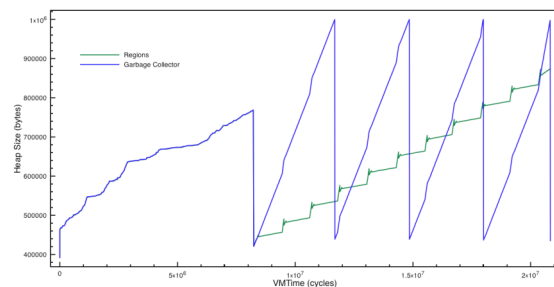
Does my method mutate the heap ?

Results :

Memory occupancy for two programs using a **Garbage Collector** or using **Regions**



- In this program, most regions are short-lived
- The program runs in nearly constant space
- No more need for a Garbage Collector



- This program uses a large mutating data structure
- Some of the generated garbage stays forever in the long-lived region
- Running this program without a GC may cause a memory leak

How to predict the runtime behaviour ?

Achieved :

- We propose a simple static analysis and allocation policy that groups data structures in regions
- Automatic region-based memory management can allow programs to run without a Garbage Collector

Perspectives :

- This approach has a tendency to place too many objects in the same region
- We need to find an algorithm to predict at compile time the behaviour of the region allocator