Designing a Concurrent Hardware Garbage Collector for Small Embedded Systems

Flavius Gruian and Zoran Salcic

...

{f.gruian,z.salcic}@auckland.ac.nz
Dept. of Electrical and Computer Engineering
The University of Auckland

10th Asia-Pacific Computer Systems Architecture Conference

Outline

1. Introduction
   - Motivation and Goals
   - Approach
2. The Garbage Collection Algorithm
   - Mark-Compact GC
   - Object and Address Space Organisation
3. Garbage Collection Unit Design
   - Architecture
   - Operation
   - Evaluation on a Test Platform
   - Adding Processor Support
   - Evaluation on the Target Platform
4. Summary & Conclusions

Motivation and Goals

Memory management in low-resource embedded systems:

- simplistic at best
- mainly software, lacking hardware support
- hardly real-time

Our goal – *designing a Garbage Collector*:

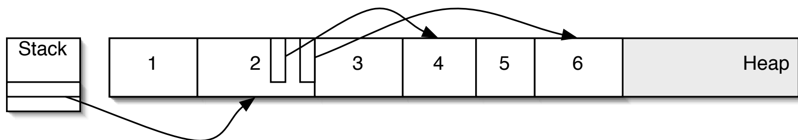
- for the Java Optimised Processor (JOP)
  - 3+1 stages pipelined stack machine
  - microprogrammed
  - direct execution of bytecodes
- hardware accelerated
- concurrent, low latency
- real-time capable

Approach

Step-by-step design and test:

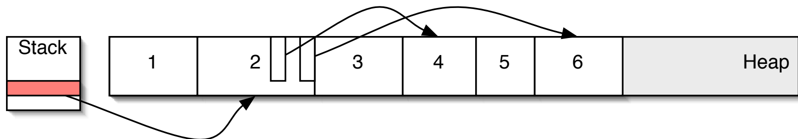
1. Software, Stop-the-World, for JOP
   - algorithm and structures check
   - executable image generator check
2. Hardware, on a Test Platform (MicroBlaze)
   - garbage collector unit test
   - system integration check
   - comparison vs. software
3. Hardware, on the Target Platform (JOP)
   - processor modifications check
   - full system test

Mark-Compact GC



A cycle starts with a mixed heap of live and dead objects. . .

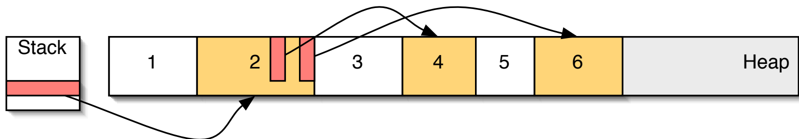
Mark-Compact GC



A cycle starts with a mixed heap of live and dead objects. . .

1. Identify and mark root references

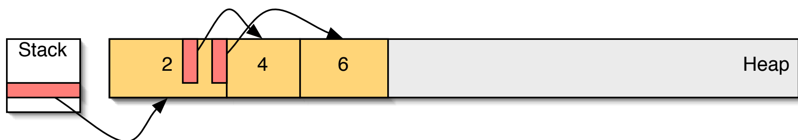
Mark-Compact GC



A cycle starts with a mixed heap of live and dead objects. . .

1. Identify and mark root references
2. Mark references pointed by already marked objects

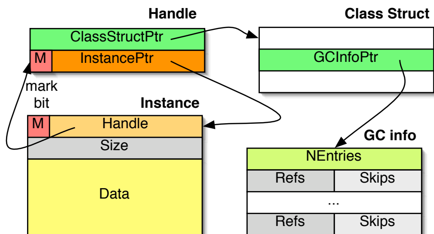
Mark-Compact GC



A cycle starts with a mixed heap of live and dead objects. . .

1. Identify and mark root references
2. Mark references pointed by already marked objects
3. Move and compact marked objects at the bottom of the heap

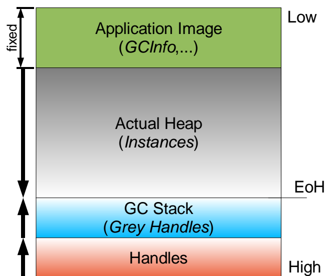
Object and Address Space Organisation



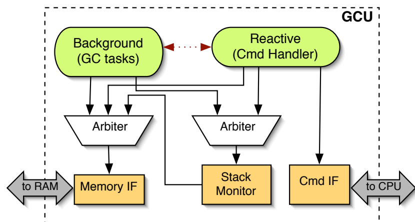
- use *handles* to easily update moving objects
- specialised structure *GCInfo* to track references inside objects
- *Marked* bit packed together with addresses

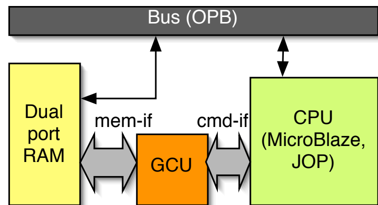
Use part of the heap for:

- object handles
- yet to scan handles (MARK phase)



Architecture





- two main processes
  - **Background**: basic GC
  - **Reactive**: commands and synchronisation handler
- shared resources
  - interfaces
  - handle stack monitor

- direct channel to CPU for commands
- direct channel to RAM for data
- dual-port RAM to offload the system bus

Operation

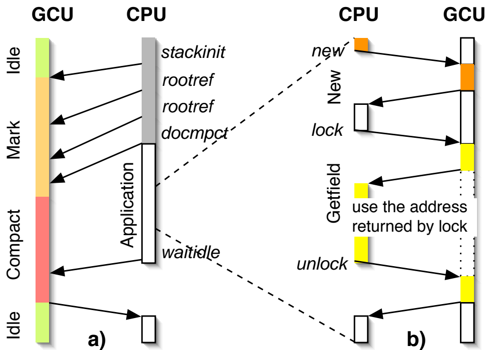
CPU view of:

A GC cycle:

1. initialize GCU
2. register root handles
3. allow COMPACT

Object access (rd/wr):

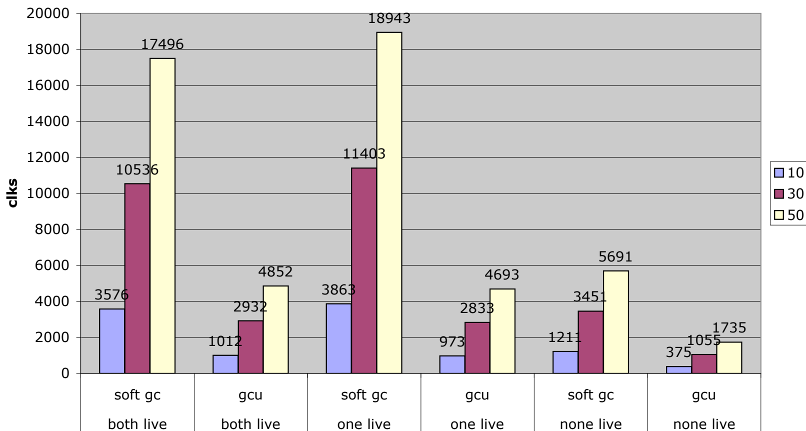
1. lock handle
2. perform operation
3. unlock handle



Timing and protocol of typical CPU-GCU interraction

Evaluation on a Test Platform

Stop-the-World on MicroBlaze with FSL channels

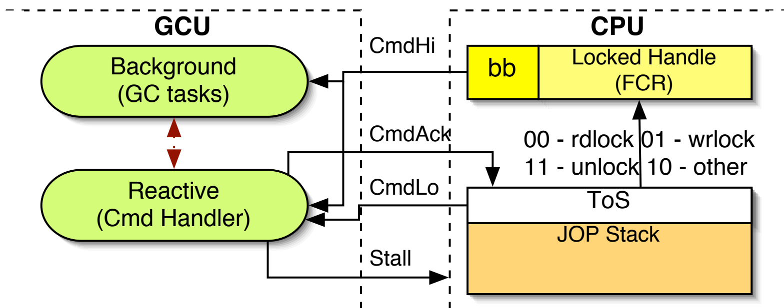


- two lists of 10, 30, and 50 elements

Adding Processor Support

Issue: Large latency of object operations ($> 19clk$):
MicroBlaze→FSL→Reactive→Background→*back*

JOP solution: Dedicated register, visible from GCU,
New dedicated load/store μinstructions,
Use *stall* instead of *ACK*



Evaluation on the Target Platform

After complete implementation of the GCU-JOP system on a Xilinx Spartan2e 600:

- Resource utilisation

Unit resources	GCU only	JOP only	**Full system resources**	JOP, RAM, IPs	JOP, GCU RAM, IPs
Slice FF	900	400	Slices	1543 (22%)	3053 (44%)
4LUT	2966	1783	BRAMs	71 (98%)	71 (98%)

- Synchronisation latency

read access bytecodes				**write access bytecodes**			
class	latency (clock cycles)			class	latency (clock cycles)		
	before	after	change		before	after	change
GEFIELD	28	31	11%	PUTFIELD	30	45	50%
*ALOAD	41	44	7%	*ASTORE	45	60	33%
ARRAYLEN	15	18	20%	NEW, *NEWARRAY	Java methods		< 1%
INVOKE*	> 100	+3	< 3%				

Summary & Conclusions

- design of a low interference, concurrent hardware GCU
- gradual design and test to handle complexity
- processor modifications for high GCU-CPU integration
- suitable for real-time systems, especially as time-triggered GC

- some issues remain ... (see the discussion in the paper)