

GENDP

AN EFFICIENT SYSTEM LEVEL GARBAGE COLLECTION FOR JAVA OS

Kai LU* Ian Rogers⁺

* School of Computer Science in NUDT

⁺ Senior Software Engineer, Azul Systems
previously Research Fellow at the University of Manchester

Agenda

- Background & Motivation
- Detail of GenDP
- Evaluation
- Conclusion and Future works

1. Background & Motivation

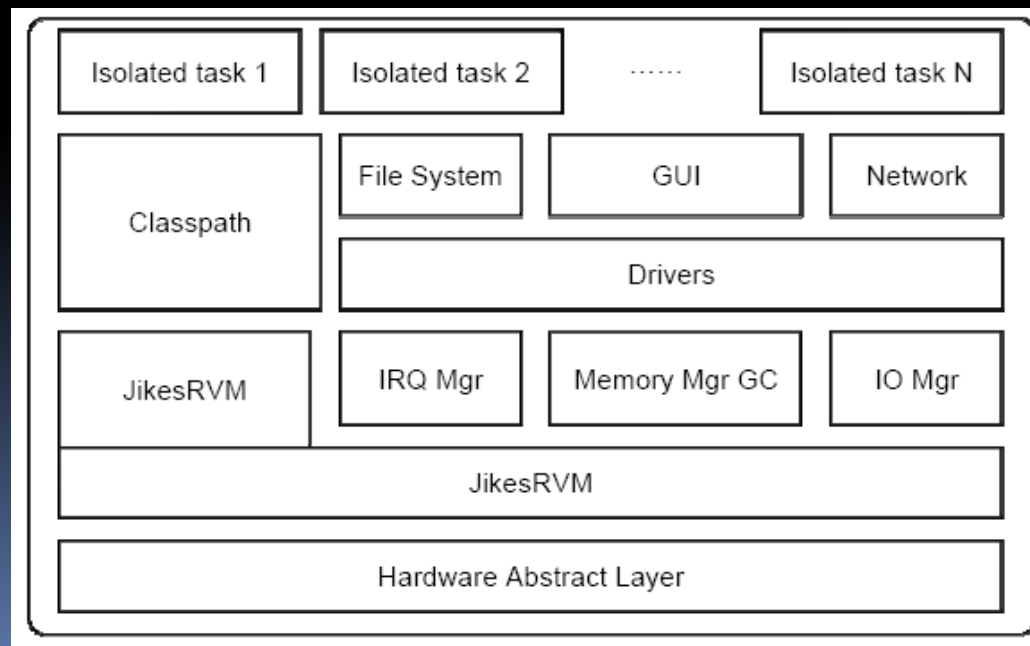
- O-O OS becomes popular now
 - JNODE: current version 0.2.8
 - Singularity: Microsoft
- How to use the OS's knowledge
- How to use the capability of hardware
 - Java OS runs at the highest privilege
 - Know more detail of hardware information
 - More powerful control ability of hardware

Background & Motivation(*cont.*)

- What is our contribution
 - Present a whole solution of Memory Management for Java OS
 - Memory Allocate
 - Utilize the dirty bit of page table as write barrier
 - Minor and major collection
 - Evaluation shows the performance improved by 10% on average

Background & Motivation(*cont.*)

- Platform is our JUnicorn OS:
 - Pure java OS
 - Micro-kernel OS
 - based on Metacircular Research Platform - MRP (formerly the JikesRVM)



Structure of Junicorn OS

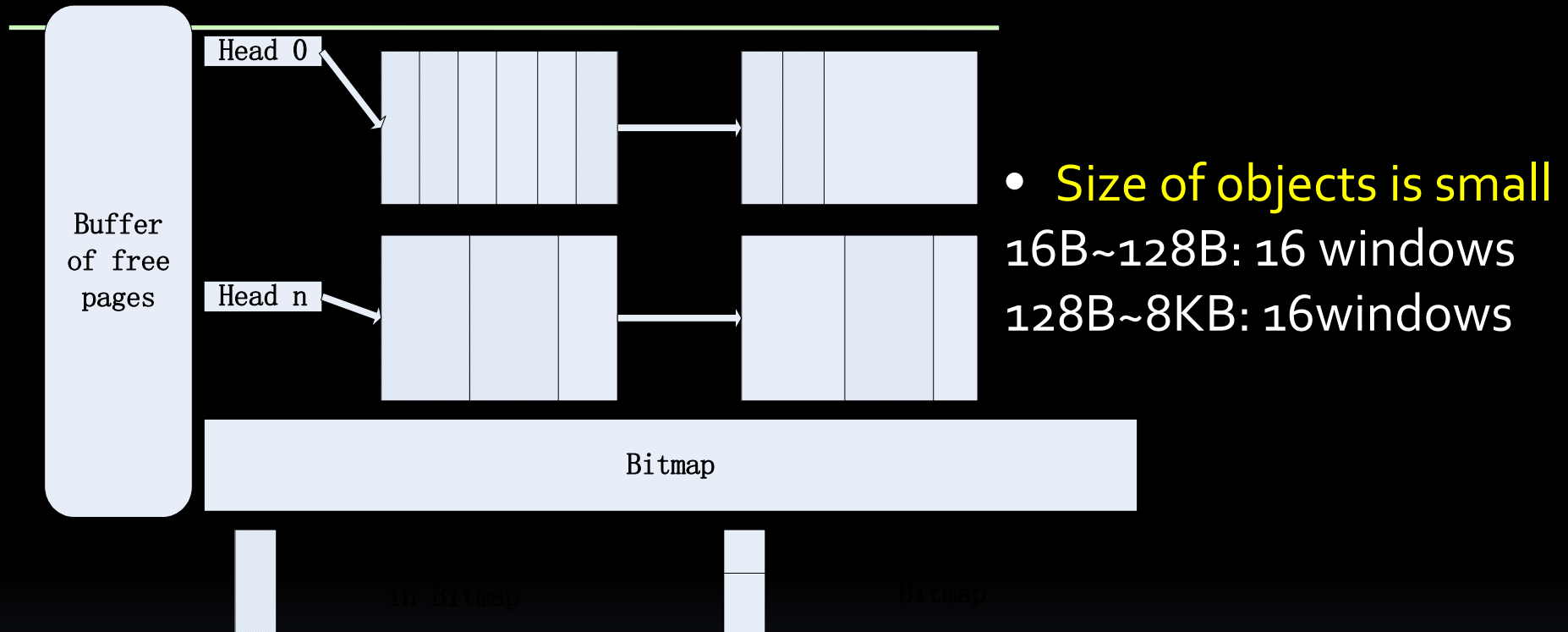
Background & Motivation(*cont.*)

- The memory management based on MMTK
 - GenMS: generational garbage collection
 - GenMS uses write barrier to intercept the write operations of inter-generation pointers
 - The overhead of write barrier is high
 - Blackburn: 6% of total running time

2.GenDP algorithm

- GenDP: Generational Dirty Page GC algorithm
- Allocation strategy of GenDP
 - Nursery space uses Bump-pointer algorithm
 - No change at all
 - Mature space uses Allocate-hole strategy
 - We use the Page table structure to manage the memory
 - Using fixed slot allocating approach

GenDP (*cont.*)



- Each kind of pages are linked in different queues
- Use the bitmap to record the status of this page
 - If the bitmap vector is ZERO, means this page is free
- Space utilization can reach to 90%

GenDP (*cont.*)

- Collection strategy of GenDP
 - Minor collection
 - GenDP: uses dirty bit of page table to find the potential inter-generation pointers
 - On X86 CPU, in hyper privilege
 - Supported by hardware, no overhead added

public static boolean isPageDirty (Address page);

public static void setPageDirty (Address page, boolean dirty);

- Scan each object in dirty page to find inter-generation pointers

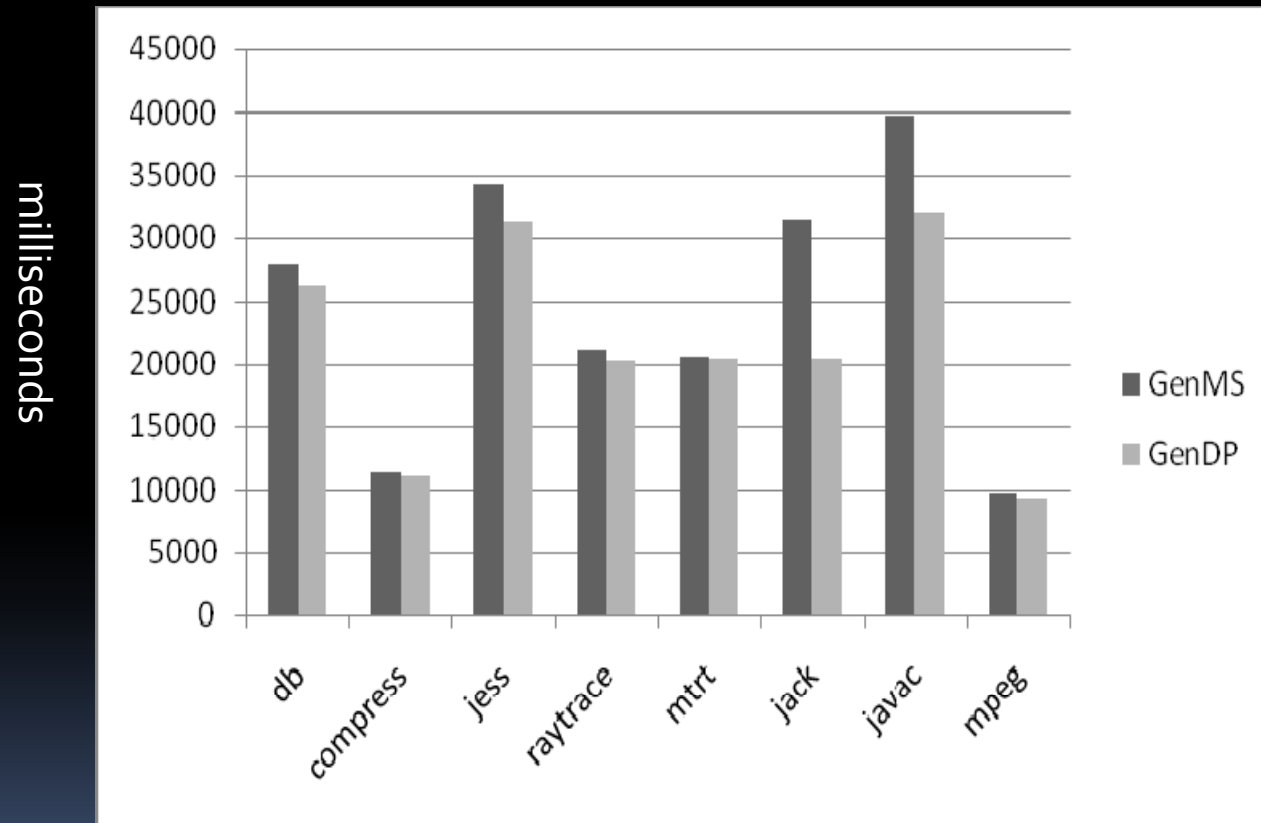
GenDP (*cont.*)

- Major collection
 - Checking alive objects from root
 - Mark the bitmap to 1 instead of 0 in object's head
 - We can allocate the free slot to new objects
 - When bitmap of a page is ZERO, we can recycle this page

Evaluation

- Platform:
 - DualCore Intel Core 2 Duo, 2200 MHz (11 x 200) CPU; 2GB of physical memory;
 - Heap size is 256MB
 - Using SPEC JVM98 benchmark
- Compared with GenMS
 - the write barrier in the GenMS uses remembered sets
 - the mature space in the GenMS is organized using segregated free-lists.

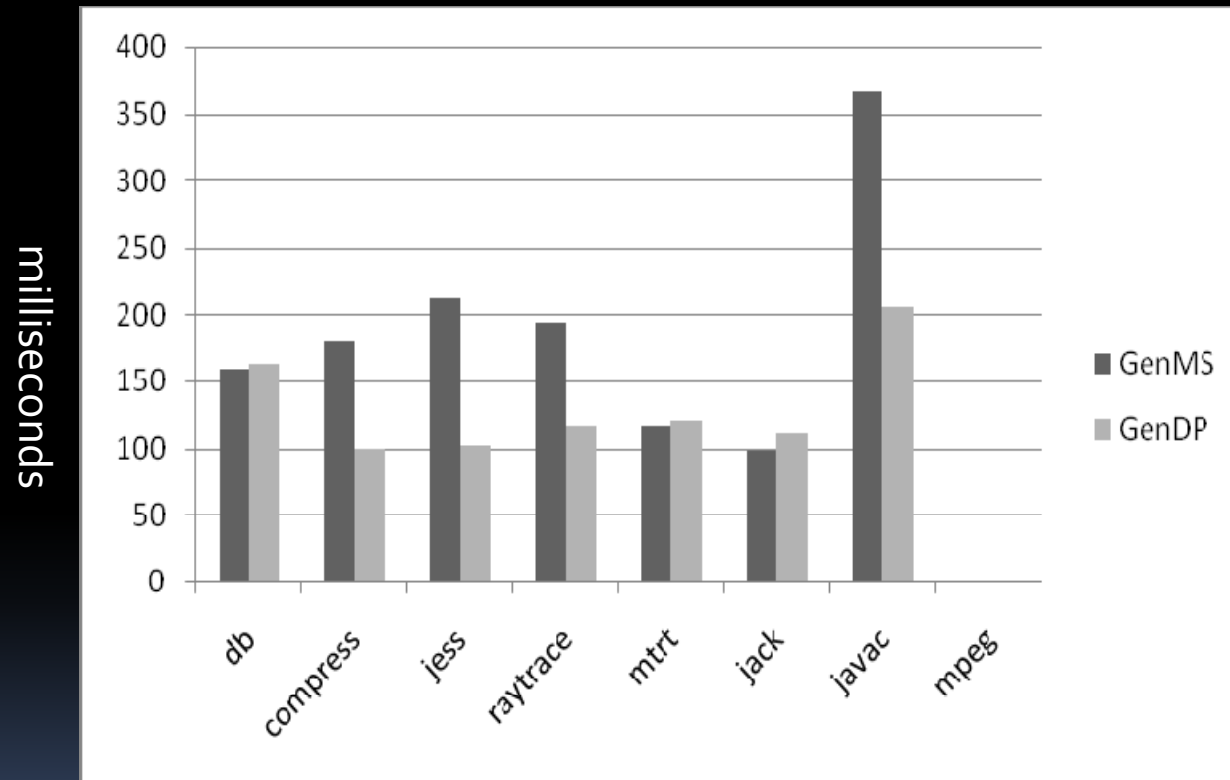
• Mutator time



- Average 10% improvement

Evaluation(*cont.*)

- Pause time



- Average 20% improvement

- Repeated times of write barrier in SSB of GenMS

Program in Spec	Number of repeated times
db	22.23
compress	202.04
jess	4.67
raytrace	5.92
mtrt	9.23
Jack	59.12
Javac	1.76
mpeg	7.12

-
- Only about 30% of pages are dirty
 - We only need to scan objects in these pages, thus we can save time

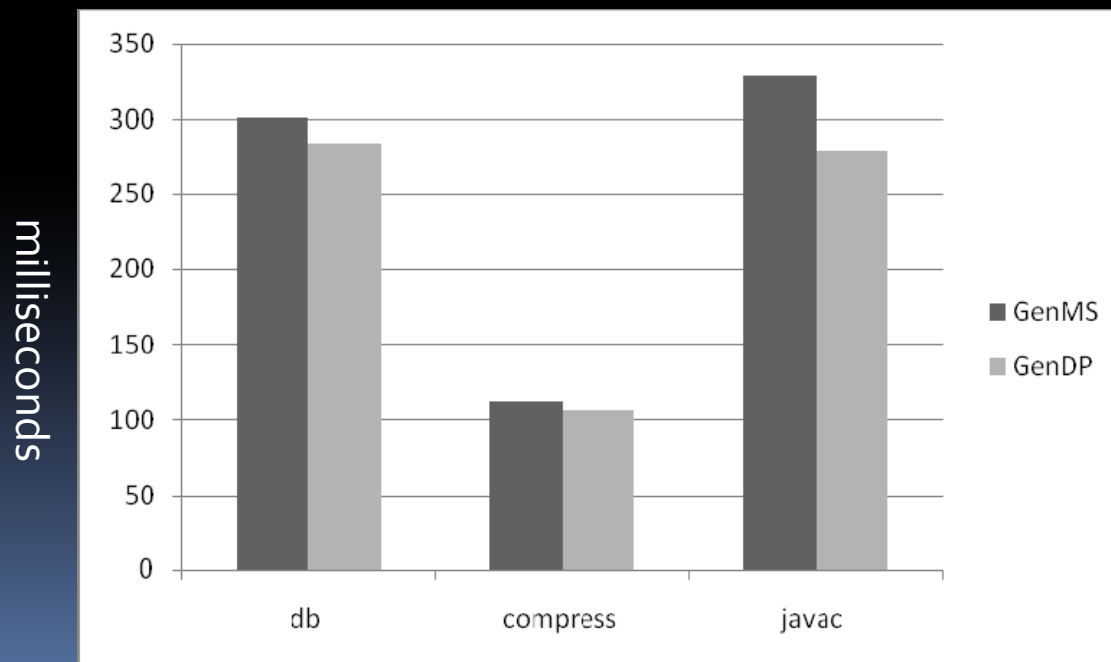
Spec programs	Ratio of dirty pages
db	35%
compress	19%
jess	6%
raytrace	10%
mtrt	6%
Jack	16%
Javac	16%
mpeg	35%

On average, GenDP's pause time is less than that of GenMS by 78%.

Evaluation(*cont.*)

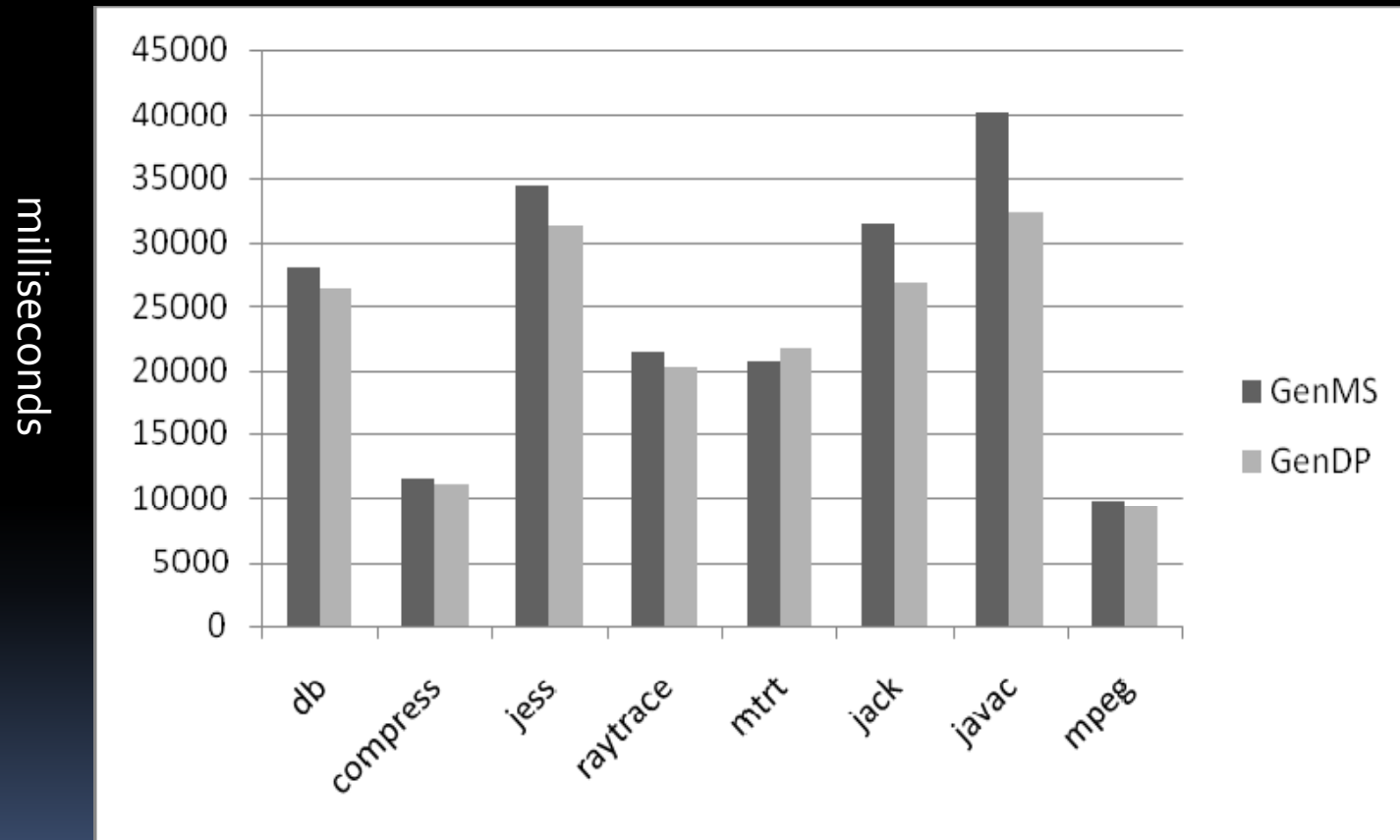
- Major collection time

benchmark	GenMS	GenDP
db	301.2	284.075
compress	112.013	106.5
javac	328.613	279.025



Evaluation(*cont.*)

- Total running time



average GenDP is 90% of GenMS

Conclusions and Future work

- As GenDP focuses on saving mutator time a lot
 - Waste Pause time a little
 - But the total running time is still better than GenMS

Maybe not suitable to all kinds of applications

- Parallel GC to improve the performance of GC further

Questions?