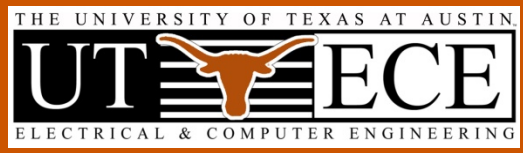


2008 IEEE International Symposium on Workload Characterization

On the Representativeness of Embedded Java Benchmarks

Ciji Isen & Lizy John - University of Texas at Austin

Jung Pil Choi & Hyo Jung Song - Samsung Electronics

Motivation

- **Ubiquitous computing – PDAs, cell phones (blackberries & iPhone), iPod, TiVo**
- **Java**
  - Increased complexity => Abstraction
  - Security – less danger of dangling pointers
  - Portability across ISAs and architectures.
  - Quick churn rate.

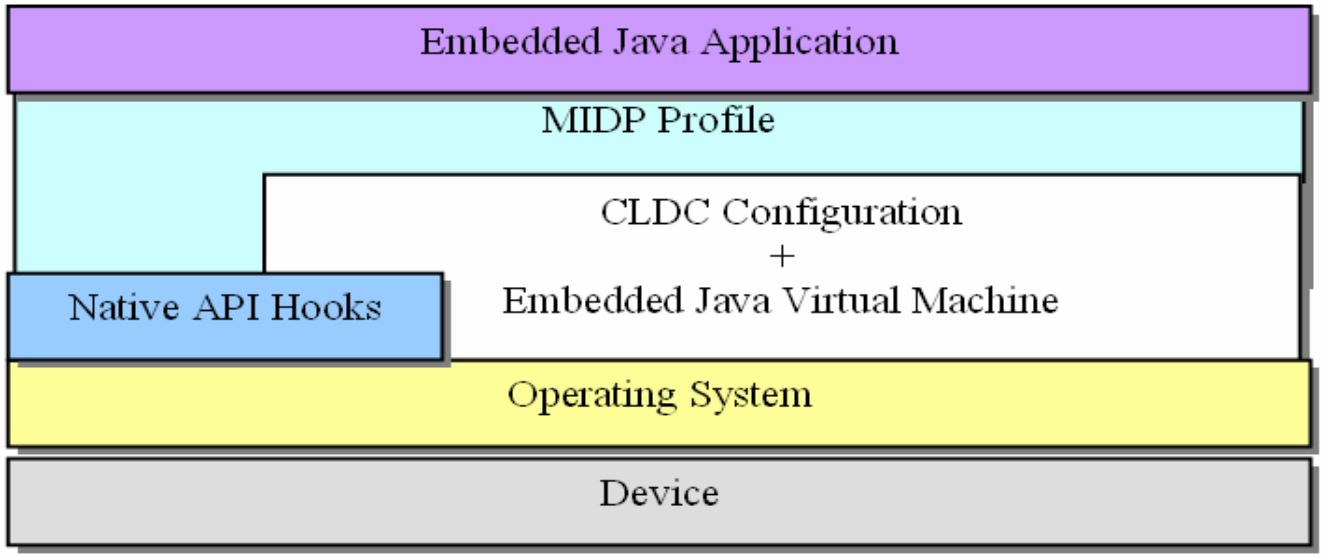
Motivation

- **Uniqueness of Embedded Java application and VM**
  - JITs and HotSpots don't work => interpretation
  - Long running applications rare
- **Very few studies on embedded Java**
  - Better benchmarks => better analysis => better design decisions

Trivia

- **Java enabled cell phones - 789 million units in 2007, 1.17 billion in 2010** ***[Gartner 2006]***
- **At least five mobile architectures in large corporation** ***[Gartner 2006]***
- **A third of current application will be discarded by 2009** ***[Gartner 2006]***

Layers in embedded Java.



Objective

- **Benchmarks - very important - but how representative are popular embedded Java benchmarks?**
- **Do embedded benchmarks differ from desktop/client side Java benchmarks?**

Embedded Java

- **Embedded Java benchmarks**
- **52 real world applications (games, browser, graphics etc)**
- **ARM optimized VM from Samsung**
- **IBM J9**

Benchmark	*Source*
EmbeddedCaffeineMark 3.0	Pendragon Software
MIDPMark	Digital Bridges
Morpmark	Morpheme
GrinderBench	EEMBC

TestName	*Description*
Chess	Chess playing solver (3 games, 10 moves)
Crypto	Encrypts/Decrypts a small text document with a set of crypto algorithms (DES, IDEA, Blowfish, Twofish)
XML	Parsing and manipulation of a small XML document
Parallel	Mergesort, Matrix multiply using multiple threads for execution
PNG	Decodes a PNG graphic image (doesn't use graphical display, just the decoding only)
RegEx	Parses a file using regular expressions

Desktop Java benchmarks

- **SPEC jvm98**
- **DaCapo**

Experimental setup: Tool chain

- ***ARM* family – ARM 9**
- **CLDC KVM 1.03 optimized by Samsung**
- **Sprint Wireless Toolkit**
- **JVMPI,JVMTI – instrumentation interface to Java**
- **IBM J9**

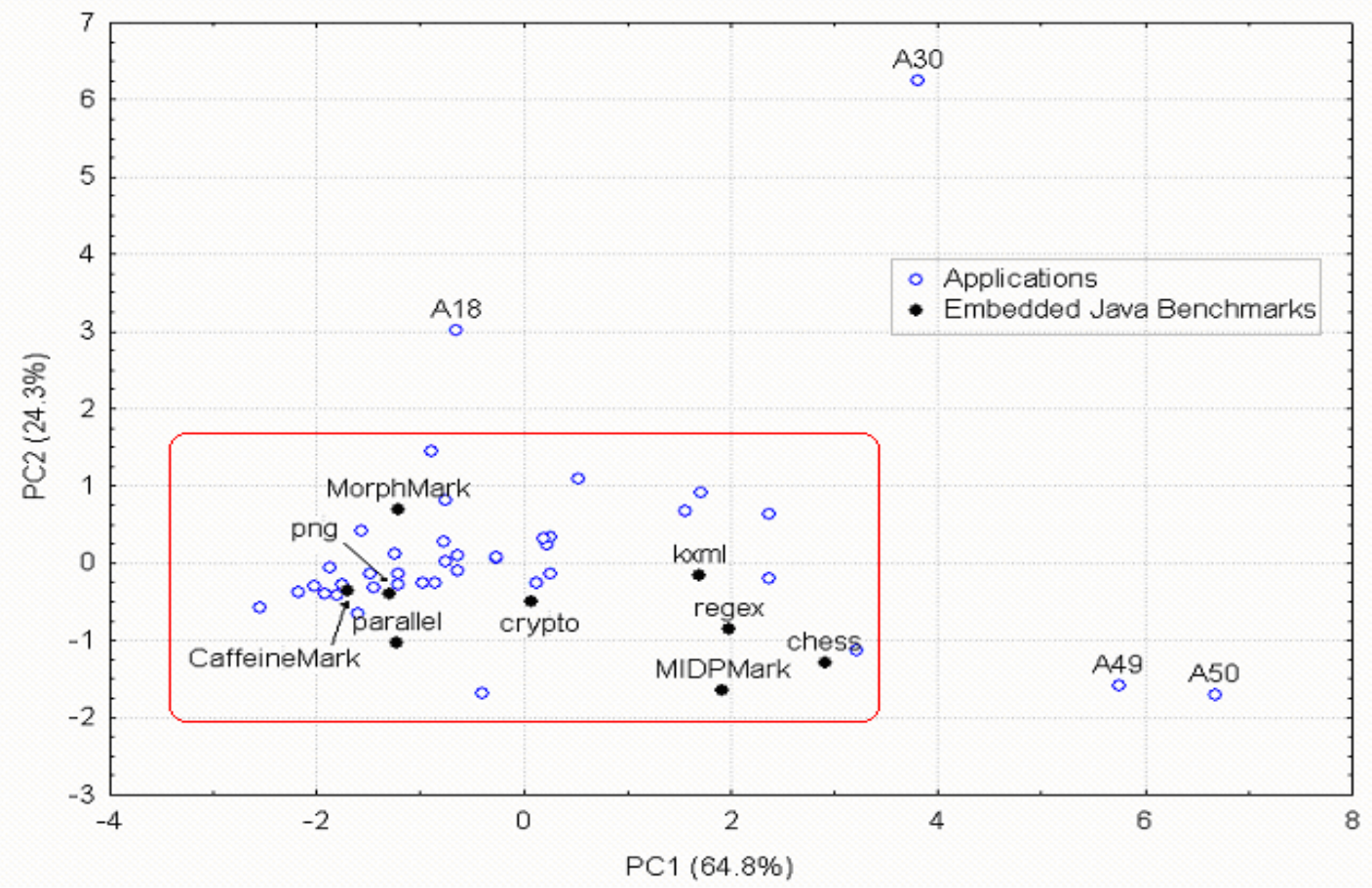
Evaluation Process

- **Collect metrics – 3 categories**
  - Code Complexity, Object management, Code reuse
- **Use Principal Component Analysis to reduce the dimension space**

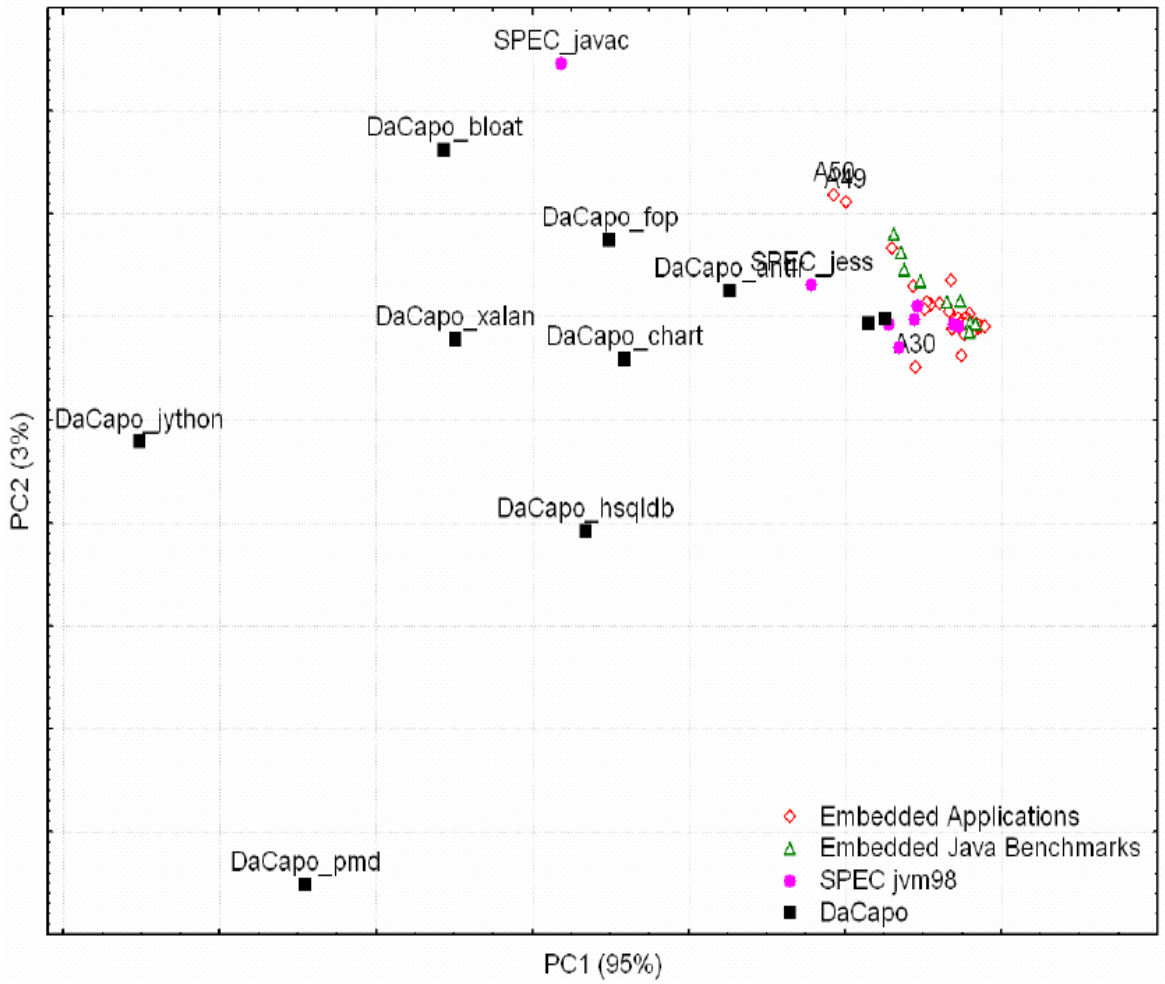
Code Complexity – CK metrics [Chidamber et al]

- **WMC (Weighted Methods per Class)**
- **DIT (Depth of Inheritance Tree)**
- **NOC (Number of Children)**
- **CBO (Coupling Between Objects)**
- **RFC (Response for a Class)**
- **LCOM (Lack of Cohesion)**

Code Complexity – Embedded Java



Code Complexity – Desktop & Embedded Java



Object Allocation/Liveness/Locality

- **Heap Volume**
  - Allocated, Live, Allocated/Live
- **Object Count**
  - Allocated, Live, Allocated/Live
- **Object Size**
  - Allocated, Live
- **Measured on IBM J9 using JVMPI – forced GC once in 10k allocation**

Object Allocation/Liveness/Locality

	Heap Volume(MB)			Average Object Size	
	Allocated	**Live**	**Alloc/Live**	**Allocated**	**Live**
Benchmarks					
Min	0.06	0.05	1.29	29.67	31.77
Max	101.11	25.62	153.82	142.08	883.32
Average	36.47	7.22	30.57	70.48	220.37
Applications					
Min	0.20	0.09	2.30	32.03	35.12
Max	211.15	0.52	407.31	49.67	90.67
Average	10.12	0.21	29.36	38.01	54.10

Object Allocation/Liveness/Locality

	Heap Volume(MB)			Average Object Size	
	Allocated	**Live**	**Alloc/Live**	**Allocated**	**Live**
Benchmarks					
Min	0.06	0.05	1.29	29.67	31.77
Max	101.11	25.62	153.82	142.08	883.32
Average	36.47	7.22	30.57	70.48	220.37
Applications					
Min	0.20	0.09	2.30	32.03	35.12
Max	211.15	0.52	407.31	49.67	90.67
Average	10.12	0.21	29.36	38.01	54.10

Code Reuse - metrics

- **Hot function call**
- **Library % of hot function calls**
- **Hot lib function calls**
- **% of calls that are to lib**
- **same metrics for cycles**
- **Total calls**
- **Total function count**
- **Measured using Spring Wireless Toolkit**

Code Reuse

	hot fn - 80% calls	hot fn - 90% calls	lib % of hot fn calls	lib % of hot fn cycles	hot lib fns - 90% cycles	% of lib cycles	Total Fnts
Benchmarks							
Min	3	4	0	0	0	0.34	78
Max	12	19	81.9	49.38	8	48.61	172
Avg	7	11	31.25	14.14	2	14.7	133
Application							
Min	1	1	2.33	6.59	1	10.94	43
Max	29	44	100	100	16	92.41	318
Avg	12	17	40.19	65.48	6	63.39	122

Code Reuse

	hot fn - 80% calls	hot fn - 90% calls	lib % of hot fn calls	lib % of hot fn cycles	hot lib fns - 90% cycles	% of lib cycles	Total Fnts
Benchmarks							
Min	3	4	0	0	0	0.34	78
Max	12	19	81.9	49.38	8	48.61	172
Avg	7	11	31.25	14.14	2	14.7	133
Application							
Min	1	1	2.33	6.59	1	10.94	43
Max	29	44	100	100	16	92.41	318
Avg	12	17	40.19	65.48	6	63.39	122

Code Reuse

	hot fn - 80% calls	hot fn - 90% calls	lib % of hot fn calls	lib % of hot fn cycles	hot lib fns - 90% cycles	% of lib cycles	Total Fnts
Benchmarks							
Min	3	4	0	0	0	0.34	78
Max	12	19	81.9	49.38	8	48.61	172
Avg	7	11	31.25	14.14	2	14.7	133
Application							
Min	1	1	2.33	6.59	1	10.94	43
Max	29	44	100	100	16	92.41	318
Avg	12	17	40.19	65.48	6	63.39	122

Summary

- **Characterized industry embedded Java benchmarks**
- **Representativeness wrt to 50 real cell phone apps**
  - Code complexity, object allocation, code reuse
- **Embedded Java benchmarks need improvement**
  - Apps have larger range vs. benchmarks
  - Stark difference in object allocation/live properties
  - Limited code reuse in Apps
- **Significant complexity of desktop Java benchmarks => Embedded Java benchmarks are distinct enough**

Acknowledgment & Q|A

- **Prof. Kathryn S McKinley – UT Austin**
- **LCA website- http://lca.ece.utexas.edu/**

- **Q|A**