Cork: Dynamic Memory Leak Detection for Garbage-Collected Languages

Maria Jump
& Kathryn McKinley

Presented by Yuhao Zhu

CS 395T

Motivation

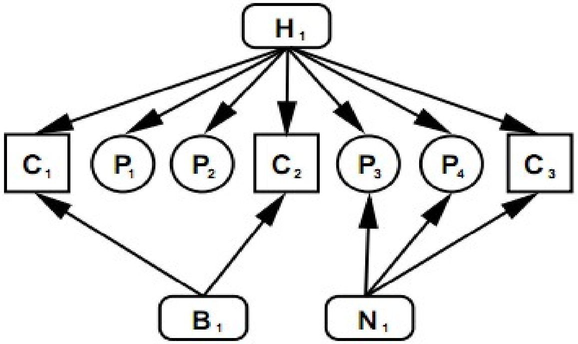
- Garbage collection has to be conservative
  - Which leads to maintaing references to inactive objs
  - Difficult to find these bugs
- Memory leak detection
  - Efficiency
  - Precision
  - Easy to parse
- Now we have Cork!

Type Points-From Graph

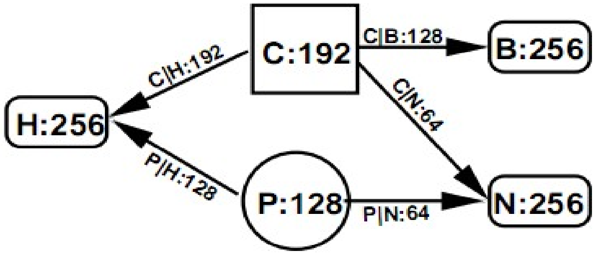
- Node: total volumn of type *t*
- Edge: t'->t where an obj of type t is pointing to obj of type t'
- Constructed during whole heap collection (scanning phase)
- Additional type lookup (upon scanning an obj)

Type	Symbol	Size
HashTable	H	256
Queue	N	256
Queue	B	256
Company	C	64
People	P	32

(a) Object statistics



(b) Object points-to graph



Detecting heap growth

- Basic idea: differencing consecutive TPFGs, but needs to consider fluctuations
- Ratio Ranking Technique:
  - Decay factor (f):

$$V_{T_i} > (1 - f) * V_{T_{i-1}}$$

  - Phase growth factor (g): Q > 1

$$g_{t_i} = p_{t_i} * (Q - 1)$$

  - Ranking by accumulating *g* over several collections, rewarding growth and penalizing decay
  - Objs with rank > $R_{threshold}$ are considered as candidates

Memory leak localization

- Data structures
  - Slice: a path on which the ranks of all edges are positive
  - The slice contains the dynamic data structures containing candidate nodes
- Allocation site:
  - Reports all allocation sites for candidates
  - Associate a *SiteMap* of a particular type with each allocation site duing compilation
  - Seach the maps to find allocation sites for a type

Optimizations

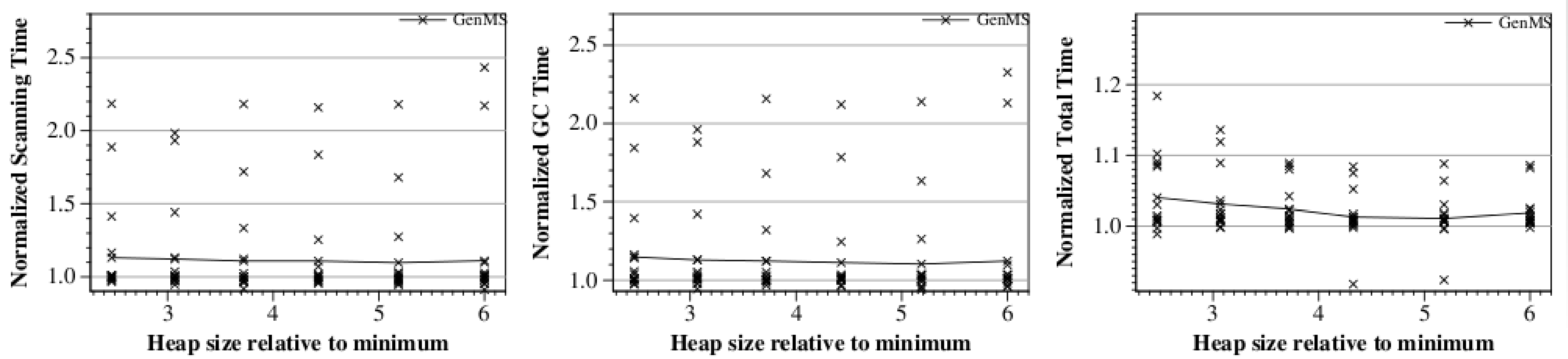
- Efficiency
  - Only look at four recent TPFGs
  - Each type has its own type information (over four collections) recorded in its Type Information Block
- Scalability
  - Edges are linear with respect to the nodes (quadratic in theory)
  - Remove edges from the edge pool and adding to the edge lists

Evaluation

	(a) Benchmark Statistics					(b) Type Points-From Statistics							(c) Space Overhead				
	Alloc	# of Colltn		# of types		# of types		# edges per type		# edges per TPFG		% pru-ned	TIB		TIB+Cork		
Benchmark	MB	whl	gen	bm	+VM	avg	max	avg	max	avg	max		MB	%H	MB	%H	Diff
Eclipse	3839	73	11	1773	3365	667	775	2	203	4090	7585	42.2	0.53	0.011	0.70	0.015	0.167
fop	137	9	0	700	2292	423	435	3	406	1559	2623	45.2	0.36	0.160	0.55	0.655	0.495
pmd	518	36	1	340	1932	360	415	3	121	967	1297	66.0	0.30	0.031	0.44	0.186	0.155
ps	470	89	0	188	1780	314	317	2	93	813	824	66.3	0.28	0.029	0.39	0.082	0.053
javac	192	15	0	161	1753	347	378	3	99	1118	2126	45.8	0.28	0.071	0.43	0.222	0.151
jython	341	39	0	157	1749	351	368	2	114	928	940	66.2	0.28	0.041	0.39	0.112	0.071
jess	268	41	0	152	1744	318	319	2	89	844	861	66.0	0.27	0.049	0.38	0.143	0.094
antlr	793	119	6	112	1704	320	356	2	123	860	1398	55.8	0.27	0.016	0.39	0.282	0.266
bloat	710	29	5	71	1663	345	347	2	101	892	1329	50.6	0.26	0.017	0.38	0.064	0.047
jbb2000	**	**	**	71	1663	318	319	2	110	904	1122	59.0	0.26	**	0.38	**	**
jack	279	47	0	61	1653	309	318	2	107	838	878	66.2	0.26	0.042	0.37	0.131	0.089
mtrt	142	17	0	37	1629	307	307	2	91	820	1047	57.5	0.26	0.081	0.37	0.258	0.177
raytrace	135	20	0	36	1628	305	306	2	91	814	1074	56.1	0.26	0.085	0.37	0.272	0.187
compress	106	6	3	16	1608	286	288	2	89	763	898	60.9	0.25	0.105	0.36	0.336	0.231
db	75	8	0	8	1600	289	289	2	91	773	787	66.1	0.25	0.160	0.35	0.467	0.307
Geomean	303	27	n/a	104	1813	342	357	2	116	1000	1303	57.4	0.29	0.048	0.41	0.168	0.145

- 44% types are resident at a time
- More than half of the edges are pruned (rank < 0)
- Very very little space overhead

Evaluation (cont.)



(a) Scan Time (b) GC Time (c) Total Time

- 11.1% to 13.2% for scan time, 12.3% to 14.9% for collector time, resulting in 1.9% to 4.0% for total time

Evaluation (cont.)

- Accuracy (sensitivity) analysis
  - The larger the better, but at some point it stops

Benchmark	(a) Decay Factor 0%	15%	25%	(b) Rank Threshold 0	100	200
Eclipse bug#115789	0	**6**	6	12	**6**	6
fop	2	**2**	2	35	**2**	1
pmd	0	**0**	0	11	**0**	0
ps	0	**0**	0	3	**0**	0
javac	0	**0**	0	71	**0**	0
jython	0	**0**	1	3	**0**	0
jess	0	**1**	2	9	**1**	1
antlr	0	**0**	0	9	**0**	0
bloat	0	**0**	0	33	**0**	0
jbb2000	0	**4**	4	10	**4**	4
jack	0	**0**	0	9	**0**	0
mtrt	0	**0**	0	3	**0**	0
raytrace	0	**0**	0	4	**0**	0
compress	0	**0**	0	4	**0**	0
db	0	**0**	0	2	**0**	0

Table 2. Number of types reported in at least 25% of garbage collection reports: (a) Varying the *decay factor* from Ratio Ranking Technique ($R^t_{thres} = 100$). We choose a decay factor $f = 15\%$. (b) Varying the *rank threshold* from Ratio Ranking Technique ($f = 15\%$). We choose rank threshold $R^t_{thres} = 100$.

Evaluation (cont.)

- Two case studies: SPECjbb2000 and Eclipse
- Find the candidates -> correlate to the code -> localize the bug
  - Human ingenuity is still required!

Discussion

- How to deal with unmanaged languages where type information is not contained?
- How to further reduce the need for programmers' instrumentations?