Cache optimizations in Garbage Collection

Bachelor Thesis

David Himmelstrup
Department of Computer Science

Research aims

1. Reasonable implementation complexity?
2. Improved performance?
3. Modern, fully-featured collectors?

Memory management

Top 10 programming languages (GitHub 2017):

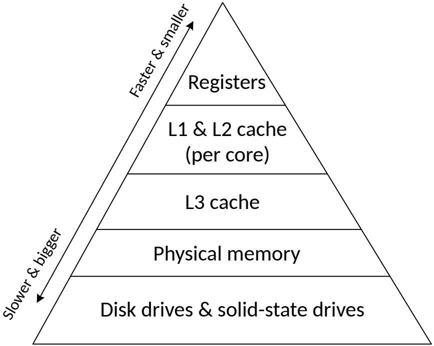
1. Javascript
2. Python
3. Java
4. Ruby
5. PHP
6. C++
7. C#
8. Go
9. C
10. Swift

Memory management

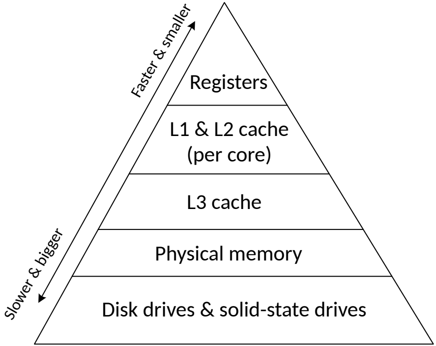
Top 10 programming languages (GitHub 2017):

1. Javascript
2. Python
3. Java
4. Ruby
5. PHP
6. C++
7. C#
8. Go
9. C
10. Swift

Memory hierarchy

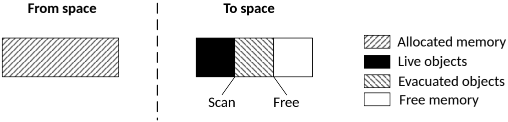


Memory hierarchy

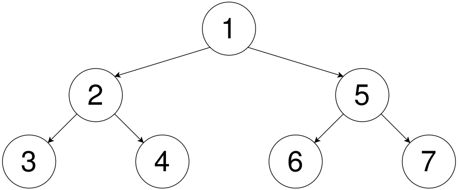


RAM: Random-Access Memory?

Semi-space garbage collection

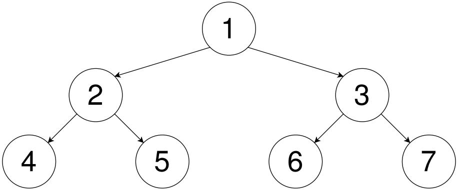


Depth-first copying

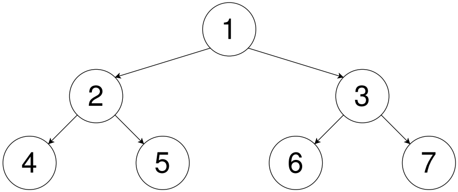


Each branch lives in continuous memory.

Breadth-first copying



Breadth-first copying

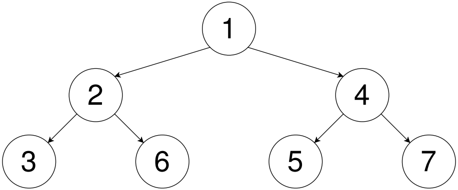


No stack but poor locality of reference.

A middle ground: tail-first copying

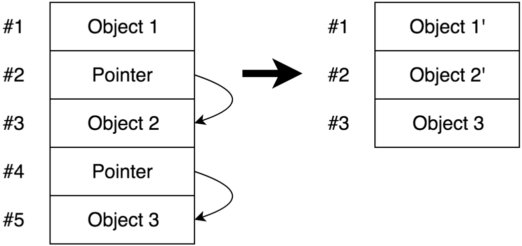
1. No stack
2. No reserved heap
3. Puts branches in *chunks* of continuous memory

A middle ground: tail-first copying



1. No stack
2. No reserved heap
3. Puts branches in *chunks* of continuous memory

New opportunity: tail compaction



Results

LLC Load Misses (millions):			
Name	**Baseline**	**Tail-copy**	**Tail-compact**
Allocate	3.9	+5.5%	+6.2%
BinTree	60.9	-4.5%	-4.9%
Bush	33.5	+4.1%	+2.8%
BushTail	50.9	+0.8%	+0.6%
Imbalanced	40.9	-10.3%	-11.0%
MemBench	84.1	-80.3%	-80.0%
PowerSet	130.3	-78.7%	-78.7%
Min		-80.3%	-80.0%
Max		5.5%	6.2%
Mean		-23.3%	-23.6%

- Mostly fewer LLC misses (-80% to +6%)

Results

Branch Misses (millions):			
Name	**Baseline**	**Tail-copy**	**Tail-compact**
Allocate	3.7	-6.1%	-13.8%
BinTree	29.0	+107.5%	+102.5%
Bush	17.4	+78.3%	+77.0%
BushTail	16.2	+44.4%	+44.2%
Imbalanced	39.4	+6.7%	+4.3%
MemBench	10.7	-7.9%	-12.8%
PowerSet	20.0	-6.7%	-10.7%
Min		-7.9%	-13.8%
Max		107.5%	102.5%
Mean		30.9%	27.3%

- Mostly fewer LLC misses (-80% to +6%)
- More complex algorithm $\Rightarrow$ more branch misses

Results

Heap size:			
Name	**Baseline**	**Tail-copy**	**Tail-compact**
Allocate	1.8 gb	+0.0%	-8.3%
BinTree	1.8 gb	+0.0%	-6.2%
Bush	1.3 gb	+0.0%	-3.1%
BushTail	5.1 gb	+0.0%	-1.7%
Imbalanced	1.6 gb	+0.0%	-6.2%
MemBench	6.7 gb	+0.0%	-5.0%
PowerSet	10.4 gb	+0.0%	-5.0%
Min		0.0%	-8.3%
Max		0.0%	-1.7%
Mean		0.0%	-5.0%

- Mostly fewer LLC misses (-80% to +6%)
- More complex algorithm $\Rightarrow$ more branch misses
- 5.0% smaller heap on average

Results

GC times:			
Name	**Baseline**	**Tail-copy**	**Tail-compact**
Allocate	1.5s	+14.5%	+3.1%
BinTree	4.6s	+16.2%	+10.2%
Bush	2.8s	+12.5%	+9.6%
BushTail	3.6s	+10.7%	+5.9%
Imbalanced	4.0s	-0.7%	-6.2%
MemBench	7.8s	-27.5%	-31.4%
PowerSet	12.6s	-31.1%	-34.5%
Min		-31.1%	-34.5%
Max		16.2%	10.2%
Mean		-0.8%	-6.2%

- Mostly fewer LLC misses (-80% to +6%)
- More complex algorithm $\Rightarrow$ more branch misses
- 5.0% smaller heap on average
- Four programs are slower and three are faster.

Conclusion

1. Reasonable implementation complexity?
   Yes.
2. Improved performance?
   Mixed performance.
3. Modern, fully-featured collectors?
   Unlikely.