NC STATE

MobiCom 2023

DroidPerf: Profiling Memory Objects on Android Devices

Bolun Li, **Qidong Zhao**, Shuyin Jiao, Xu Liu

NC State University

October 5, 2023

Motivations

Existing popular profilers on ART (Android runtime)

- *PowerTutor, PowerScope* - Energy consumption analysis
- *Android profiler* - Hotspot analysis
- *Perfetto* - Trace analysis
- *Other profilers* - Cross-layer inefficiency or app crash analysis

Why we need a profiler on ART (Android runtime)

- Existing tools mainly focus on hotspot analysis profiling
- Existing tools fail to tell whether a resource is being used productively and contributes to a program's overall efficiencies
- Android applications are highly susceptible to object locality issues

Need a new profiler for **profiling memory objects** on Android devices

Study of Memory Inefficiencies in Android Applications

Android unique memory inefficiency causes

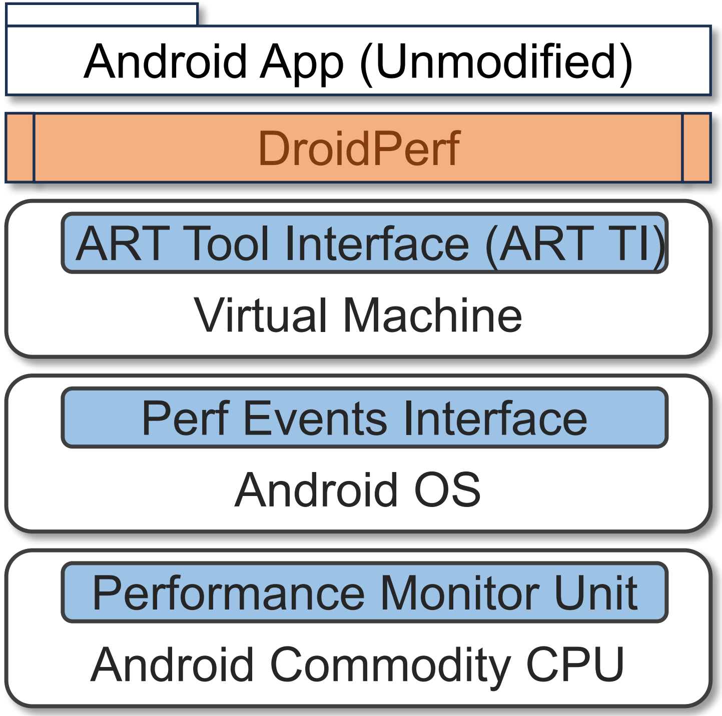
- Memory Bloat - High memory usage in loops
- Android persistent running services
- Improper use of Android APIs
  e.g., Android broadcast APIs
- Runtime configuration changes
  e.g., screen orientation changes
- Use Android unoptimized data structure
  e.g., HashMap (ArrayMap is optimized)
- Generate a large amount of duplicated objects

Challenges of Building an Android Memory Profiler

- ❑ Lack of library to support hook the object allocation on Android
- ❑ Limited JVM Tool Interface (JVMTI) support
  - ❑ Missing important callbacks, e.g., callbacks when a method is compiled and loaded into memory, or unloaded from memory
- ❑ Lack of async unwind facility
  - ❑ Missing AsyncGetCallTrace facility, which is used to get calling stack inside a signal handler to avoid the safe point
- ❑ Unable to obtain sampled PMU effective address on ARM
  - ❑ ARM only provides an instruction pointer on each PMU sample

Methodology

DroidPerf in system stacks



Methodology

Capture object creation and destruction

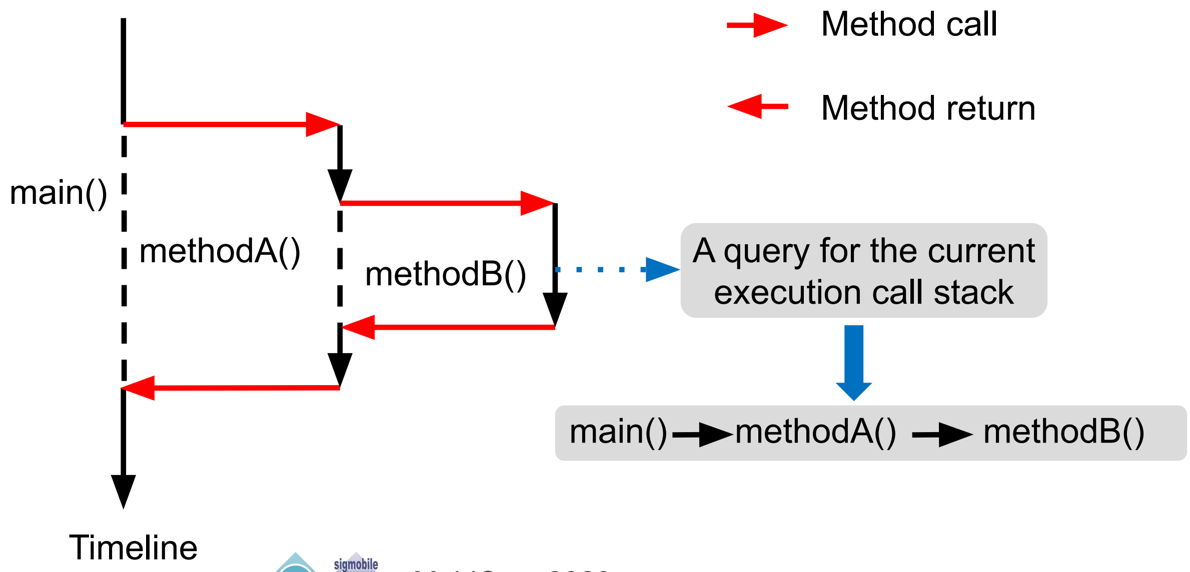
- **JVMTI** ObjectAlloc Callback
  - Allocated thread
  - Reference to the allocated object
  - Reference to the class of the allocated object
  - Object Size
- **JVMTI** ObjectFree Callback
  - Tag of the freed object

Methodology

Construct call path

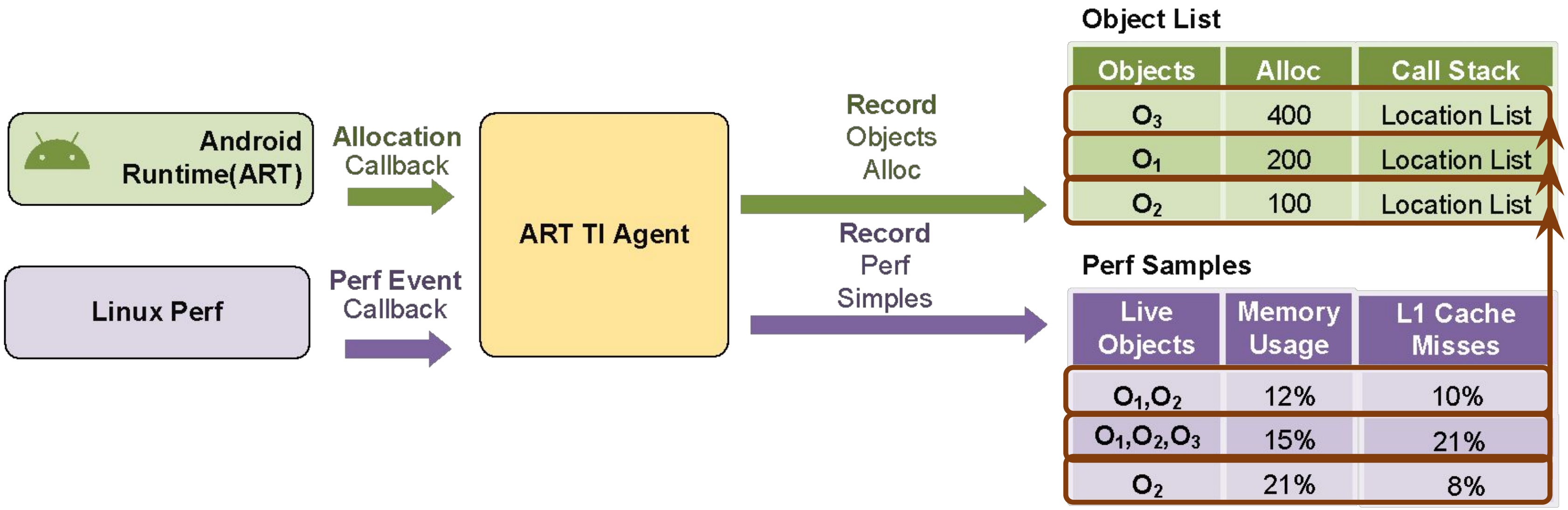
- Monitor Executed Function: **MethodEntry & MethodExit**
- Obtain line number: **GetLineNumberTable** API

```
main() {
  methodA();
}
methodA() {
  methodB();
}
```



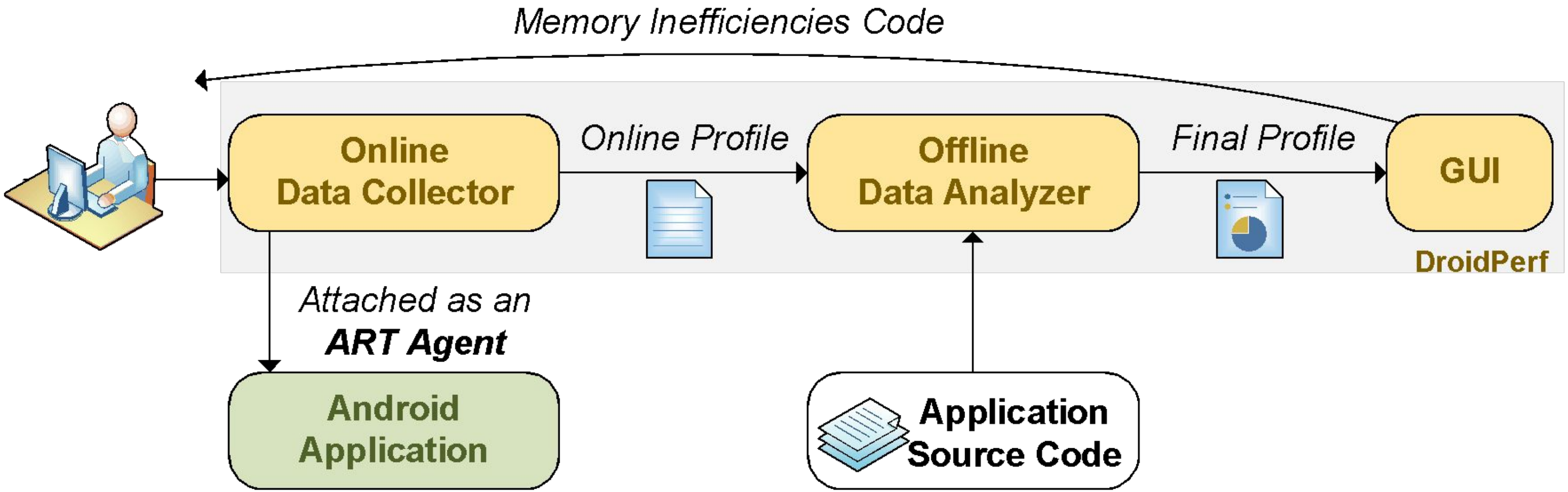
Methodology

DroidPerf's object-centric analysis



Overview of DroidPerf

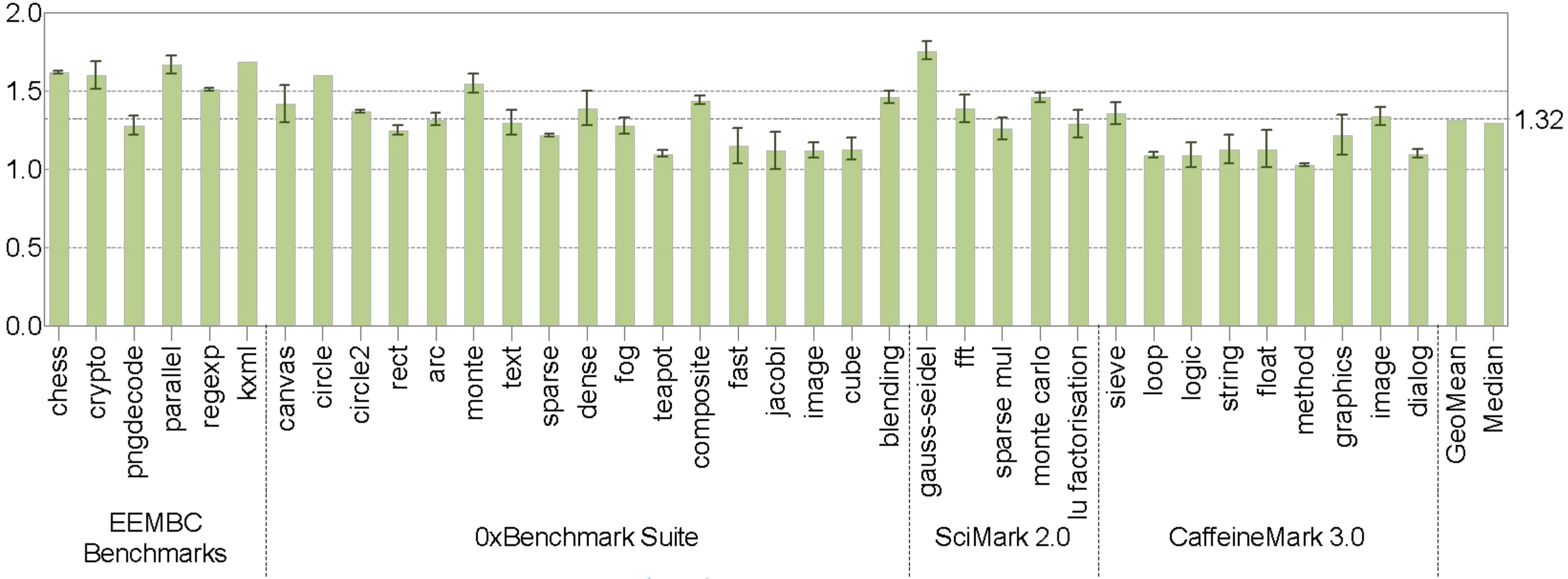
Workflow of DriodPerf



Evaluation

Time overhead

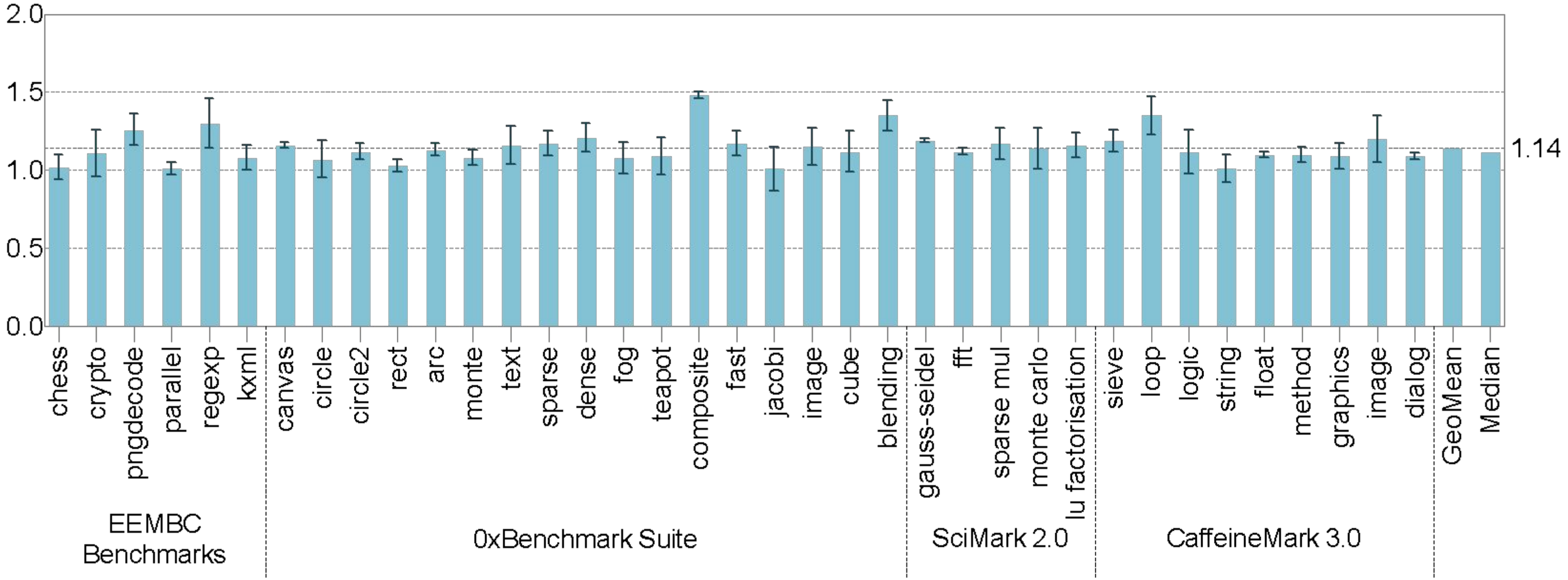
- Google Pixel 7 (**Google Tensor G2 chipset, 8 GB RAM**)



Evaluation

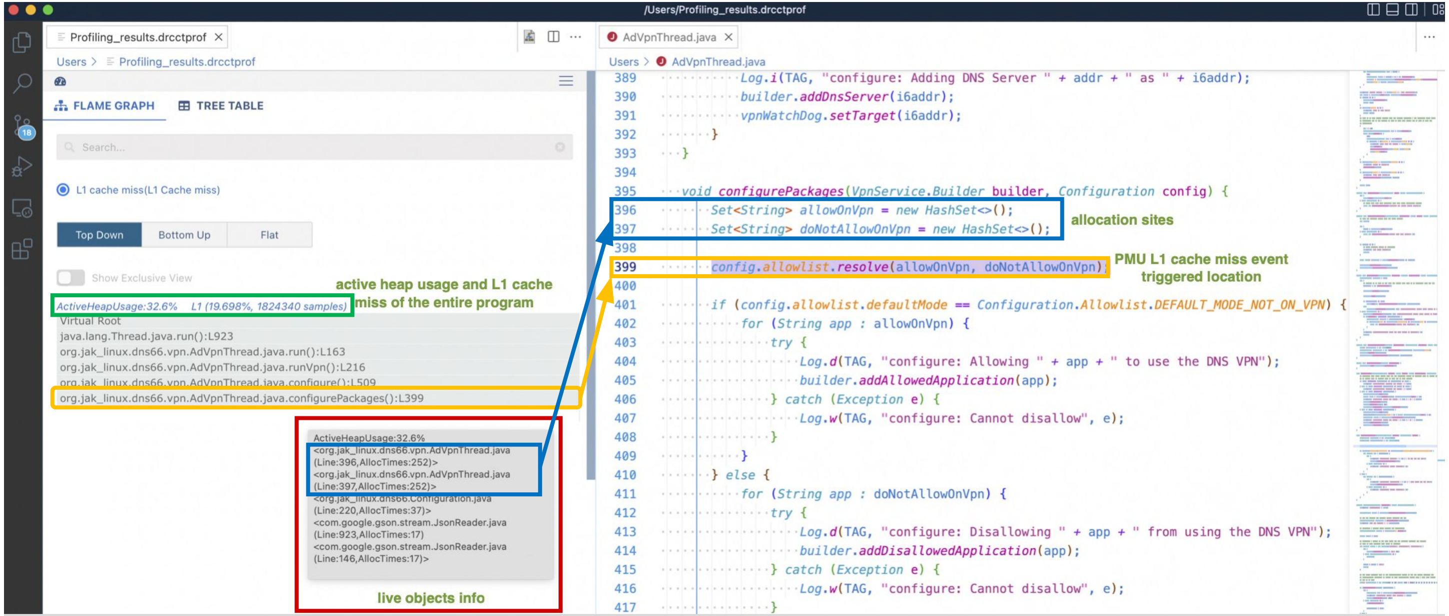
Memory overhead

- Google Pixel 7 (**Google Tensor G2 chipset, 8 GB RAM**)



Case Study of DNS66

VSCode GUI on Android Application DNS66



Optimization result: reduce the total cache misses by 18.97% and the heap memory usage consumption by 6.6%

Case Studies

Android Applications	Inefficiency		Optimization			
	Problematic Code	*Type*	*Patch*	*WH*	*WCM*	*WEI*
DNS66	AdVpnThread.java	Excessive memory usage in nested loops	Move problematic allocation sites out of loop and reset them upon request	18.97%	6.60%	10.62%
Rajawali	RenderToTextureFragment.java			15.69%	7.57%	6.37%
RxAndroid	HandlerScheduler.java			9.67%	5.48%	3.94%
Applozic	MessageClientService.java	Long running services	Avoid using persistent services	17.94%	8.16%	9.36%
GmsCore	TriggerReceiver.java	Inefficient using Android broadcast API	Unregister the broadcast receiver which is no longer needed	8.89%	5.56%	4.98%
Twire	Service.java TopStreamsActivity.java	Runtime configuration (screen orientation, etc.) changes	Reuse the visual elements	7.25%	5.37%	14.14%
Stream Chat	QueryChannelsLogic.java			7.24%	6.57%	11.01%
Stetho	AsyncPrettyPrinterRegistry.java ChromeDevtoolsServer.java JsonRpcPeer.java ResponseBodyFileManager.java	Poor data structure	Use Android optimized data structure	6.29%	13.43%	5.01%
MediaPicker	Utility.java	Generate duplicated objects	Reuse an object that memorized from the last used point	12.88%	4.06%	8.13%
LeakCanary	AndroidDebugHeapAnalyzer.java			9.49%	3.93%	7.14%
Foxy Droid	QueryBuilder.java			8.55%	1.51%	3.92%

Many optimization patches were confirmed: *Applozic, GmsCore, Twire, etc.*
Upstreamed optimization patches: *DNS66, Rajawali*

Conclusions

- Categorize different **Android memory inefficiencies**
- Develop **DroidPerf** to pinpoint object-level inefficiencies that occurred on Android Runtime
  - Support lightweight novel object-centric profiling
  - Apply to unmodified applications
  - Obtain nontrivial performance gains
- Contribute to the community
  - **Many optimization patches** are upstreamed to real-world applications

Future Work

- Enhance DroidPerf's Feature
  - Support instruction-level monitoring
  - Support instruction-level inefficiency analysis
- Enhance DroidPerf's Usability
  - Develop an extension for Android Studio to integrate DroidPerf into the Android Development Environment