UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

JOSÉ ROBERTO LEIVA HÉRCULES

Análise de Impacto da Máquina Virtual ART em Sistemas Android

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Antonio Carlos S. Beck
Co-orientador: Anderson L. Sartor

Porto Alegre
2016

AGRADECIMENTOS

A apresentação deste trabalho significa o fechamento de uma fase muito importante da minha vida, a qual começou sete anos atrás, quando decidi sair do meu país de origem (Guatemala) em busca de uma formação de melhor qualidade da que eu poderia encontrar nele.

Durante este trajeto tive a oportunidade de ter ao meu lado pessoas e órgãos que de alguma forma facilitaram ou fizeram mais ameno o processo, e aos quais dedico este capítulo.

Primeiramente quero agradecer à minha família, que me serve de inspiração e sempre me apoia e motiva para alcançar meus objetivos. Meus pais, Antonio e Lizeth, que aceitaram a ideia de ficarmos longe para que eu conseguisse uma formação de qualidade, e especialmente por todo o esforço e dedicação que tiveram comigo e meus irmãos desde o nascimento de cada um de nós. Meus irmãos, Andrea e Tony, com os quais sempre foram trocadas ideias que possibilitassem o desenvolvimento das atividades. Quero também deixar registrado um agradecimento especial ao Tony, quem se formou pela UFRGS como engenheiro de computação em 2014/2, e foi a pessoa que mais me ajudou nesta trajetória, desde que me recebeu aqui no Brasil.

Gostaria de agradecer ao MEC e aos governos do Brasil e da Guatemala por terem o convênio PEC-G, que me possibilitou a participação nesta grande experiência. Agradeço à UFRGS e seus professores, por sua grande qualidade de ensino e pela infraestrutura da instituição, que possibilita a qualquer aluno a aprender e realizar os seus trabalhos.

Agradeço ao meu orientador e meu co-orientador deste trabalho, professor Antonio Beck e Anderson Sartor, os quais foram de suma importância pelo apoio, suporte e orientação durante toda a realização deste trabalho.

Finalmente quero agradecer à minha namorada Malu, que me acompanhou durante estes últimos quatro anos de faculdade estando sempre do meu lado, e à sua família (família Coronel) pelo apoio dado durante todo este tempo. Assim, agradeço aos colegas com quem dividi as tarefas a serem realizadas nos trabalhos da faculdade, em especial a Juliano, que foi com quem realizei mais destes trabalhos.

A todos os envolvidos no meu crescimento acadêmico e pessoal nesta instituição, meu muito obrigado!

RESUMO

Atualmente, Android é o Sistema Operacional (SO) para portáteis mais popular no Brasil e no mundo. No mundo está presente em aproximadamente 70,84% dos dispositivos móveis e *tablets*, enquanto que no Brasil esse número aumenta para 85,03%. Desde seu início, o Android utilizou a máquina virtual (VM) Dalvik como intermediário entre a linguagem a nível de aplicação e a linguagem a nível de máquina. Mas em 2014, a Google lançou a ART (*Android Runtime*) como máquina virtual padrão para este SO, substituindo a Dalvik a partir da versão 5.0 (Lollipop). Esta mudança foi feita devido a esta última apresentar limitações em termos de desempenho e consumo de energia, a causa disto é a camada adicional que interpreta o código Java para a arquitetura alvo. Neste trabalho apresentamos a extensão da ferramenta de *profiling* multi-arquitetural para aplicações Android, AndroProf. Esta suporta as arquiteturas ARM, MIPS e x86 para a VM Dalvik. A extensão foi focada em portar os mecanismos desenvolvidos no AndroProf para suportar a máquina virtual ART em arquiteturas ARM. A extensão deste *profiler* é apresentada e uma análise comparativa entre as máquinas virtuais ART e Dalvik é realizada. Para tal, um conjunto de *benchmarks* composto por aplicações desenvolvidas puramente em Java e aplicações que utilizam código nativo é usado para a avaliação de desempenho e consumo de energia dessas máquinas virtuais.

Palavras-chave: Android. Máquina Virtual. Dalvik. ART. Profiling. Desempenho. Consumo de energia.

Impact Analysis of ART Virtual Machine in Android Systems

ABSTRACT

Currently, Android is the most popular mobile Operating System (OS) in Brazil and worldwide. It is present in about 70.84% of the mobile devices and tablets in the world, whereas in Brazil this number increases to 85.03%. Since its inception, Android has used the Dalvik virtual machine as an intermediary between an application level language and machine level language. But in 2014, Google launched ART (Android Runtime) as a standard VM (Virtual Machine) for this OS, replacing Dalvik from version 5.0 (Lollipop). This change was made due to the latter presents limitations in terms of performance and energy consumption. The cause of this its the additional layer that interprets the Java code to the target architecture. In this work we present the extension of the multi-architectural profiling tool for Android applications, AndroProf. It supports the ARM, MIPS and x86 architectures for Dalvik VM. The extension was focused on porting the mechanisms developed in AndroProf to support the ART virtual machine in ARM architectures. The extension of this profiler is presented and a comparative analysis between the virtual machines ART and Dalvik is performed. To this end, a set of benchmarks composed of applications developed purely in Java and applications that use native code is used to evaluate the performance and power consumption of these virtual machines..

Keywords: Android. Virtual Machine. Dalvik. ART. Profiling. Performance. Energy consumption.

LISTA DE FIGURAS

Figura 2.1 – Camadas da Arquitetura Android....14
Figura 4.1 – Funcionamento do AndroProf para Dalvik....23
Figura 4.2 – Funcionamento do AndroProf para ART....25
Figura 4.3 – Tradução binária, extensão do AndroProf para suportar ART....26
Figura 4.4 – Fluxo do QEMU modificado....27
Figura 4.5 – AndroProf Instruction Categorization GUI....29
Figura 4.6 – AndroProf Analysis GUI....30
Figura 6.1 – Razão de instruções Dalvik/ART....35
Figura 6.2 – Tempo estimado de execução em segundos....37
Figura 6.3 – Consumo de energia....38
Figura 6.4 – Razão de instruções Java/JNI....39
Figura 6.5 – Tempo estimado de execução em segundos....40
Figura 6.6 – Consumo de energia....40

LISTA DE TABELAS

Tabela 3.1 – Características dos Profilers apresentados 19
Tabela 5.1 – Conjuntos de Benchmarks 32
Tabela 6.1 – Informações de potência e de ciclos para ARM 36

LISTA DE ABREVIATURAS E SIGLAS

3G	Third Generation mobile phone network
AES	Advanced Encryption Standard
AOT	Ahead Of Time
API	Application Programming Interface
ARM	Advanced Risk Machine
ART	Android Runtime
AVD	Android Virtual Device
BB	Basic Block
CPI	Cycles Per Instruction
CPU	Central Processing Unit
FFT	Fast Fourier Transform
GCC	GNU Compiler Collection
GPS	Global Positioning System
GUI	Graphical User Interface
ISA	Instruction Set Architecture
JIT	Just In Time
JNI	Java Native Interface
JRE	Java Runtime Environment
JVM	Java Virtual Machine
LCD	Liquid Crystal Display
LU	Lower Upper (fatorização linear de matrizes)
MIPS	Million Instruction Per Second
NDK	Native Development Kit
SO	Sistema Operativo
PID	Process Identifier
QEMU	Quick Emulator
RAM	Random Access Memory
RSA	Rivest-Shamir-Adleman (criptografia)
SDK	Software Development Kit
SSH	Secure Shell
TB	Translated Block
UFRGS	Universidade Federal do Rio Grande do Sul

Wi-Fi	(trademark) Wireless Fidelity
x86	Processador Intel 32-bits
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO 11
2 CONCEITOS E DIREÇÕES INICIAIS 13
2.1 Profiler 13
2.2 Android 13
2.3 Dalvik 14
2.4 ART 15
2.5 QEMU 15
2.6 Escolha das versões 16
3 TRABALHOS RELACIONADOS 17
3.1 Profilers 17
3.1.1 Android Device Monitor 17
3.1.2 Power Tutor 18
3.1.3 GreenDroid 18
3.1.4 Relação entre Profilers e AndroProf 18
3.2 Análise sobre a máquina virtual ART 20
4 ANDROPROF PARA ART 23
4.1 Ambiente de trabalho 24
4.2 QEMU 25
4.3 Interfaces 28
4.3.1 Instruction Categorization GUI Tool 28
4.3.2 Analysis GUI Tool 30
5 BENCHMARKS E AVALIAÇÃO DA MÁQUINVA VIRTUAL ART 31
5.1 Benchmarks 31
5.2 Avaliação da Máquina Virtual ART 32
5.2.1 ART vs Dalvik executando com código Java 32
5.2.2 Java vs JNI na máquina virtual ART 33
6 RESULTADOS 34
6.1 ART vs Dalvik 34
6.1.1 Instruções executadas 34
6.1.2 Desempenho e consumo de energia 36
6.2 Java vs JNI em ART 38
6.2.1 Instruções executadas 38
6.2.2 Desempenho e consumo de energia 39
7 CONCLUSÃO 42
REFERÊNCIAS 43
APÊNDICES: TABELAS REFERENTES AO NÚMERO DE INSTRUÇÕES 45

1 INTRODUÇÃO

Atualmente, Android é o Sistema Operacional mais utilizado em dispositivos móveis no mundo. Segundo o site StatCounter (2016), aproximadamente 70,84% deles possuem este SO, enquanto que no Brasil essa porcentagem aumenta para 85,03%. Um dos motivos para este domínio no mercado é devido ao suporte multi-arquitetural, suportando as arquiteturas ARM, MIPS ou x86.

Para a criação de uma aplicação na área de Sistemas Embarcados, cada vez se faz mais importante levar em consideração tanto o desempenho quanto o consumo de energia, uma vez que existem limitações de bateria, memória e armazenamento neste tipo de dispositivo. Por isto, desenvolvedores têm criado ferramentas que possibilitam o acompanhamento do desenvolvimento de aplicações, em relação a consumo de energia e desempenho, visando auxiliar aos desenvolvedores encontrar os pontos críticos de suas aplicações, uma vez que a análise destes quesitos é necessária durante todas as etapas de criação.

Além disso, constantemente são introduzidos novos modelos de dispositivos que trazem novas especificações a serem estudadas. Tudo isto faz com que seja ainda mais complexa a criação de ferramentas que nos ajudem a analisar os benefícios e desvantagens do sistema Android.

A plataforma Android utiliza máquinas virtuais para executar/interpretar os códigos das aplicações: Dalvik e ART. Estas VMs serão apresentadas em detalhes na próxima seção, pois é de suma importância fazer uma comparação entre elas já que estão fortemente ligadas com os aspectos que serão analisados, desempenho e consumo de energia.

O objetivo do trabalho é analisar o impacto, em relação ao desempenho e ao consumo de energia, que a máquina virtual ART tem em aplicações Android. Para isto, neste trabalho estendemos a ferramenta chamada AndroProf, criada por Sartor, Correa e Beck (2013), que originalmente suporta Dalvik, mas atualmente passou a suportar também a ART. Esta ferramenta facilita a análise, uma vez que cobre um grande intervalo de informações sobre a execução de aplicações Android, permitindo ao desenvolvedor conhecer mais sobre os requisitos das aplicações.

Este trabalho visa ter uma visão diferente às encontradas nos trabalhos relacionados, apresentando uma análise que leva em consideração as aplicações por separado, para uma análise mais pontual sobre cada uma delas, assim como analisar o impacto da máquina virtual ART não só em relação à máquina virtual Dalvik, mas também em relação ao tipo da aplicação, podendo esta conter todo o código escrito em Java ou conter partes em código nativo.

Como comentado anteriormente, no segundo capítulo deste trabalho apresentaremos as máquinas virtuais. Também apresentaremos brevemente características de um *profiler*, o emulador padrão deste SO e abordaremos a nossa escolha das versões Android que foram utilizadas para a análise do impacto da máquina ART em sistemas Android. No terceiro capítulo serão apresentados os trabalhos relacionados às características analisadas neste trabalho, mostrando os *profilers* que foram utilizados nesses trabalhos e a forma com que foram desenvolvidas as análises das máquinas virtuais, as quais apresentam uma abordagem de estudo diferente à nossa, principalmente em relação aos dados coletados, uma vez que nosso *profiler* permite a análise de aplicações por separado, principal diferença em relação aos estudos feitos pelos autores dos trabalhos relacionados. O quarto capítulo abordará as características principais do *profiler* estendido, o qual mantém praticamente as mesmas características que possuía na versão que suportava somente Dalvik. Mostraremos as modificações feitas no emulador e as interfaces utilizadas para obtermos o comportamento desejado, as quais nos possibilitaram a análise feita sobre ambas as máquinas. Tendo o *profiler* pronto para uso, é necessário o uso de *benchmarks*, em forma de aplicação, que nos permitam analisar as diferenças no comportamento de ambas as máquinas virtuais. Isto será abordado no quinto capítulo, e falaremos sobre a avaliação utilizada para fazermos esta análise. O capítulo seis mostrará os resultados da análise, apresentando um conjunto de medidas que foram feitas sobre ambas as máquinas virtuais. Finalmente, no capítulo sete, falaremos sobre as conclusões obtidas após a realização deste trabalho.

2 CONCEITOS E DIREÇÕES INICIAIS

Neste capítulo apresentaremos os conceitos e as diretrizes iniciais para um bom entendimento da monografia. Começaremos falando sobre o SO Android, para conseguirmos um melhor entendimento geral, depois passaremos a falar sobre as máquinas virtuais deste SO, as quais servirão para a nossa análise, apresentando características pontuais sobre cada uma delas. Posteriormente apresentaremos o QEMU, o emulador no qual foram feitas as modificações que permitiram uma posterior análise do impacto da máquina ART em relação à Dalvik. Para finalizar o capítulo mostraremos a nossa escolha entre as versões Android para realizar este trabalho.

2.1 Profiler

É chamado de *profiler* àquela ferramenta que auxilia ao desenvolvedor a obter características de perfil e/ou os requisitos do sistema ou programa sendo desenvolvido. No caso do Android, a maioria dos *profilers* analisam as características das aplicações sendo desenvolvidas. O AndroProf, *profiler* estendido neste trabalho, possibilita a análise, em relação ao desempenho e ao consumo de energia, tanto do sistema operacional Android quanto das aplicações executadas nele, através do PID de cada processo executado pelo SO.

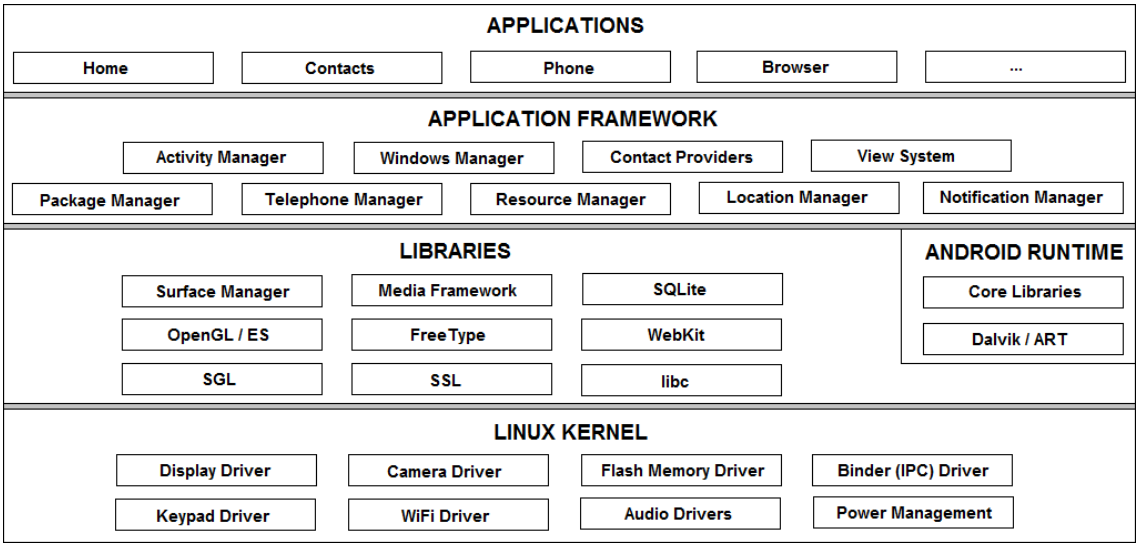
2.2 Android

Android é um sistema operacional baseado em Linux, projetado para ser utilizado em dispositivos móveis. Tem uma arquitetura em 4 camadas (Van Bockhaven, 2014), como podemos observar na Figura 2.1: camada do Kernel Linux, camada de Bibliotecas, camada de Framework para Aplicações e camada de Aplicações.

A camada do Kernel Linux é a camada mais próxima do hardware, e provê uma abstração entre o hardware e os *drivers*. A camada das bibliotecas contém bibliotecas Java específicas para desenvolvimento de aplicações Android, funcionando como interface entre o Framework e o Kernel, assim como contém as máquinas virtuais Android, as quais apresentaremos nesta seção. A camada de Framework para Aplicações provê serviços de alto-nível para as aplicações na forma de classes Java, disponibilizando para os desenvolvedores o

uso destes serviços. A camada de Aplicações é onde são encontradas e desenvolvidas todas as aplicações Android.

Figura 2.1 – Camadas da Arquitetura Android



Fonte: Van Bockhaven e Van Kerkwijk (2014, p. 10)

Código C e C++ podem também ser implementados no Android, utilizando o NDK (*Native Development Kit*) e integrando com o Java através da JNI (*Java Native Interface*). Aplicações podem conter tanto código Java quanto nativo. Porém, as máquinas virtuais Android interpretam o código Java, enquanto o código nativo não é interpretado, mas tem de ser compilado para cada arquitetura alvo, já que este não passa pela máquina virtual.

Neste trabalho, o foco principal está na camada das bibliotecas, onde são encontradas as máquinas virtuais Android. Android utiliza uma abordagem similar à utilizada pela *Java Virtual Machine* (JVM) para compilar código, convertendo o código Java *bytecode* em arquivos .dex/.efl, para Dalvik/ART respectivamente, e encapsulando-os num arquivo apk.

Máquinas virtuais nos proporcionam o benefício de poder executar a mesma aplicação em diversas arquiteturas, sem que seja necessária a recompilação do código. Porém, também existe a desvantagem de introduzir um custo adicional à aplicação, fazendo que tenha menor desempenho. A diferença entre as máquinas virtuais do Android está no modo em que a interpretação é feita, conforme veremos a seguir.

2.3 Dalvik

É a máquina virtual, criada pela Google, que foi utilizada para rodar as aplicações Android nas versões anteriores à versão Android 5.0 (Lollipop). Ela é baseada no modo de

compilação JIT (*Just In Time*), que significa que, cada vez que inicializada a aplicação ou durante execução da aplicação, a parte de código requerida será interpretada (traduzida para código de máquina) e executada.

Como, na época em que o Android foi desenvolvido, os telefones móveis tinham limitações em relação à memória RAM e ao espaço de armazenamento, compilação JIT era ideal para este cenário, visto que a Dalvik gerencia eficientemente a memória RAM, interpretando partes de código conforme o progresso da aplicação.

2.4 ART (Android Runtime)

A máquina ART foi introduzida experimentalmente para desenvolvedores na versão Android 4.4, na qual muitos deles observaram uma melhora no desempenho, mesmo estando em estado de desenvolvimento. A partir de 2014, na versão Android 5.0, ART foi lançada oficialmente como a máquina virtual padrão para sistemas Android.

Para manter retro-compatibilidade, ART utiliza o mesmo tipo de entrada *bytecode* que a Dalvik. Dessa forma, as aplicações criadas para Dalvik funcionam também na ART. ART, diferentemente da Dalvik, utiliza o modo de compilação AOT (*Ahead Of Time*) o que significa que o código é compilado quando a aplicação é instalada. A razão para mudar para ART foi para incrementar a velocidade de execução de código, introduzindo suporte melhorado para processadores multinúcleo e suporte para implementação em 64 bits (uma vez que memória e espaço de armazenamento ficaram cada vez mais baratos e mais acessíveis), possibilitando a utilização de um novo modo de compilação e otimização.

2.5 QEMU

QEMU é um simulador de código aberto que funciona a nível de instrução e emula uma plataforma virtual de hardware para diferentes arquiteturas, incluindo aquelas mais utilizadas em sistemas embarcados como: ARM, MIPS e x86. Por estas características é utilizado como base para o emulador do Android.

O suporte multi=arquitetural do QEMU é possível devido ao seu mecanismo dinâmico de tradução binária, que traduz um bloco básico de código (BB) por vez, da máquina hóspede para a hospedeira. Por causa disto, a sua velocidade de execução é mais lenta que um hardware real; também, por ser um simulador que trabalha no nível das instruções, não

modela pipeline e acessos a memória. Este tipo de simulador tem um funcionamento mais rápido que os simuladores a nível de ciclos, porém é menos preciso, uma vez que as informações obtidas estão relacionadas ao conjunto de instruções da arquitetura e não ao número de ciclos que dada instrução ou tarefa leva para ser executada.

2.6 Escolha das versões

Sabemos que ART foi introduzida experimentalmente na versão 4.4, sendo lançada oficialmente na versão 5. Portanto, todas as versões do Android que possuem a ART foram avaliadas. Para alcançar o comportamento do AndroProf é necessário fazer modificações no código fonte do QEMU disponibilizado pela Android no seu SDK. No entanto, durante o estudo das versões notamos que nas versões do SDK referentes às versões 6 e 7 do Android não é mais disponibilizado o código fonte do QEMU, o que nos impossibilitou portar o *profiler* para estas versões.

Na versão 5 nos deparamos com outra característica do SDK que não nos permitia a extensão deste *profiler* para a ART. Depois de compilado pelo modo de compilação indicado pela Android, vimos que, ao abrirmos o Gerenciador de Versões, não era disponibilizada nenhuma arquitetura para esta versão, sendo possível rodar o emulador somente com AVDs (*Android Virtual Device*) criadas por meio de outras versões do SDK. Esta versão faz uso de códigos e arquivos binários pre-compilados e disponibilizados dentro do mesmo SDK para rodar o emulador. Dessa forma as modificações feitas no código do QEMU não eram espelhadas no comportamento. Foram feitas várias tentativas de modificar o comportamento do modo de compilação padrão do SDK, assim como foram usados comandos padrão de opções do Android, as quais possibilitam o uso de AVDs criadas por meio de outras versões de SDK, mas nenhuma destas tentativas terminou bem sucedida.

Devido às características das versões mais recentes do Android, o AndroProf foi estendido para a versão 4.4 unicamente, por esta conter o código fonte do QEMU, fornecer as arquiteturas (ARM e x86), poder executar com ART e por espelhar o comportamento das modificações feitas no QEMU. Para a análise comparativa entre ambas as máquinas virtuais foram utilizadas as versões 4.0 e 4.4. A versão 4.0 uma vez que foi a versão utilizada para a criação do *profiler* AndroProf e já mostrava o funcionamento desejado.

3 TRABALHOS RELACIONADOS

Android é um sistema operacional utilizado em grande escala e é uma plataforma de código aberto (*open source*). Estas características permitem aos pesquisadores a análise da sua estrutura e organização. Com isto, uma série de trabalhos têm sido propostos para analisar o desempenho ou o consumo de energia/potência do sistema Android.

Para falar dos trabalhos relacionados dividiremos esta seção em duas partes. Na primeira subseção apresentaremos *profilers* utilizados/criados por alguns pesquisadores para análise da nova máquina virtual do Android, sendo que alguns destes pesquisadores utilizaram estes *profilers* também para fazer uma análise comparativa entre ambas as máquinas. Na segunda subseção apresentaremos brevemente os trabalhos relacionados que, além de utilizarem estes *profilers*, estão relacionados com a análise em ART ou em ambas as máquinas virtuais do Android, levando em consideração os aspectos de consumo de energia e/ou de desempenho.

3.1 Profilers

Para análise de desempenho ou de consumo de energia geralmente são utilizadas ferramentas que ajudem na visualização de algumas características e comportamentos das aplicações. Cada vez mais os desenvolvedores de aplicações fazem uso destas ferramentas, ou até mesmo desenvolvem novas ferramentas com uma funcionalidade específica.

Nesta subseção serão apresentadas ferramentas que têm sido utilizadas para analisar ART e será feita uma breve comparação entre elas e o nosso *profiler*, levando em consideração características que julgamos serem relevantes para análise.

3.1.1 Android Device Monitor (Android Device Monitor 2016)

Ferramenta utilizada por Konradsson (2015) para analisar o uso de memória. Provê uma interface para debug e análise de aplicações Android. Faz parte do SDK Android e contém um conjunto de ferramentas como Hierarchy Viewer, Traceview, Systrace e DDMS. Esta última permite a análise do uso de memória por processos.

3.1.2 PowerTutor (PowerTutor 2016)

Utilizado por (Georgiev, Sillitti e Succi 2014), é uma ferramenta que analisa o consumo de potência em aplicações Android. Permite a análise de cada aplicação separadamente e provê uma interface amigável para o desenvolvedor. Considera o consumo de potência dos componentes individualmente, analisando LCD, CPU, Wi-Fi, 3G, GPS e áudio.

3.1.3 GreenDroid

Profiler desenvolvido por Couto, Carçao, Cunha, Fernandes e Saraiva (2014), identifica blocos de código e localiza aqueles que são responsáveis por consumo de energia anormal, através de um conjunto de estatísticas coletadas e analisadas pelo *profiler*, relacionando coeficientes de consumo de energia dos componentes com o código da aplicação. Mostra o resultado final da análise numa interface, para facilitar ao desenvolvedor a análise da aplicação. Classifica os métodos do projeto em categorias de acordo com seu consumo de energia. Faz uso de outra ferramenta auxiliar, também implementada por eles, o jInst, que é uma adaptação do PowerTutor, utilizado como API, que instrumenta o código fonte da aplicação para uso do GreenDroid.

3.1.4 Relação entre Profilers e AndroProf

Para finalizar esta subseção faremos uma breve comparação entre os *profilers* apresentados anteriormente em relação ao AndroProf (detalhado no capítulo 4), levando em consideração algumas características que julgamos serem importantes para um *profiler*:

1. Profile para código Java e JNI: uma vez que existem aplicações que fazem uso de código nativo para sua execução. Nenhuma das ferramentas mostradas tem suporte para ambos os códigos, somente para Java.

2. Profile separado por aplicação: já que no desenvolvimento de uma aplicação estamos interessados em obter dados sobre aquela aplicação que estamos desenvolvendo. Das ferramentas apresentadas anteriormente Android Device Monitor e GreenDroid possuem esta característica, PowerTutor parcialmente.

3. Profile para pontos de maior consumo de energia: isto é preciso para o desenvolvedor conseguir, caso necessário, mudar o código da sua aplicação para um

algoritmo diferente que gaste menos tempo e/ou energia. Somente GreenDroid possui este tipo de profile.

4. Profile de dados ilimitados: é necessário que o *profiler* consiga processar muita informação, uma vez que aplicações podem ser executadas durante muito tempo sem serem interrompidas. Só GreenDroid faz profile de múltiplos dados (conforme espaço de armazenamento).

5. Estimar desempenho: importante para conhecer os requisitos da aplicação. Android Device Monitor cobre este ponto. PowerTutor e GreenDroid não suportam.

6. Estimar dissipação de potência e consumo de energia: essencial já que sabemos que a capacidade das baterias em sistemas embarcados é limitada. Só GreenDroid e PowerTutor fornecem este tipo de suporte.

7. Prover interfaces gráficas: interfaces gráficas trazem benefícios para a análise dos dados coletados. Todas as ferramentas apresentadas possuem interface gráfica.

8. Suporte para múltiplas arquiteturas: Já que uma aplicação Android pode ser executada em múltiplas arquiteturas (ARM, MIPS, x86). Android Device Monitor e GreenDroid possuem este tipo de suporte.

9. Suporte para múltiplos dispositivos: além de suportar múltiplas arquiteturas deve suportar múltiplas organizações de processadores, devido ao grande número de organizações diferentes existentes no mercado. Android Device Monitor e GreenDroid suportam múltiplos dispositivos, PowerTutor suporta parcialmente.

Essas nove características citadas anteriormente são características do *profiler* AndroProf, em ambas as versões, exceto o suporte multi-arquitetural na versão ART. Como podemos ver na Tabela 3.1, nenhum dos *profilers* apresentados nesta seção cobre todas as características citadas, as quais consideramos relevantes para auxílio do desenvolvedor.

Tabela 3.1 – Características dos Profilers apresentados

Profiler	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*
A. Device Monitor		X			X		X	X	X
PowerTutor		--				X	X		--
GreenDroid		X	X	X		X	X	X	X
AndroProf (Dalvik)	X	X	X	X	X	X	X	X	X
AndroProf (ART)	X	X	X	X	X	X	X		X

Legendas:		
X: Suporte completo	--: Parcialmente suportado	Em branco: Não suportado

Fonte: o autor

3.2 Análise sobre a máquina virtual ART

A seguir apresentaremos os trabalhos relacionados que utilizaram algum dos *profilers* discutidos/citados na subseção anterior e que tenham relação com os assuntos tratados neste trabalho. O primeiro trabalho está relacionado por ter criado um *profiler* para analisar o consumo de energia em aplicações Android, enquanto que os demais trabalhos estão relacionados por também compararem as duas máquinas virtuais Android em algum dos quesitos apresentados. Finalizaremos esta subseção com uma breve comparação em relação ao trabalho desenvolvido nesta monografia.

1. Detecting Anomalous Energy Consumption: Neste trabalho, feito por Couto, Carçao, Cunha, Fernandes e Saraiva (2014) os autores dizem que conforme o tempo passa, os desenvolvedores de aplicações estão mais preocupados em reduzir o consumo de energia das suas aplicações. Com este intuito eles apresentam uma ferramenta (GreenDroid) para detectar anomalias no consumo de energia em aplicações Android, relacionando-as diretamente com o código fonte, sem importar a arquitetura para a qual estejam sendo desenvolvidas. O trabalho consiste na criação de duas ferramentas. A primeira, jInst, é uma ferramenta auxiliar que é um modelo calibrado dinâmico para análise de consumo de energia. É integrado, na forma de uma classe Java, ao PowerTutor, para ser utilizado como uma API (*Application Programming Interface*) para monitorar e detectar anomalias na execução da aplicação. A segunda ferramenta, é o *profiler* GreenDroid, apresentada na subseção 3.1.3. Porém, neste trabalho não há análise comparativa entre as máquinas virtuais Android. O objetivo é analisar o consumo de energia das aplicações Android.

2. Performance Compared: A proposta do trabalho “ART and Dalvik Performance Compared” (Konradsson 2015) é comparar ambas as máquinas em relação ao tamanho da aplicação, ao uso de memória RAM e ao desempenho. O tamanho da aplicação foi comparado com 3 códigos simples e 4 aplicações reais (Blogger, Drive, Fitness e Gmail) com o método de experimentação. A memória RAM foi analisada com a ajuda do Android Device Monitor (do Android SDK). Para isto foram utilizados dois emuladores, um rodando Dalvik e o outro rodando ART, sendo que os testes foram feitos com outras seis aplicações reais (Drive, Gmail, Whatsapp, Netflix, Dropbox e Skype). Para finalizar, o desempenho foi analisado utilizando 5 *benchmarks* (Linpack, Real Pi, Quadrant, AnTuTu e CF-Bench) diferentes que foram rodados 5 vezes cada, em uma arquitetura ARM, levando em consideração a média das execuções. Um ponto importante é que os *benchmarks* e as aplicações foram executados com

o mínimo de aplicações instaladas alheias aos testes, porém, os dados coletados para esta análise consideram os dados das tarefas rodadas em segundo.

3. Performance Analysis: ao serem feitas as comparações de desempenho entre ART e Dalvik no trabalho “Performance Analysis for Android” (Yadav e Bhadoria 2015), os autores também levaram em consideração os dados das tarefas executadas em segundo plano. Os testes realizados levam em consideração unicamente o tempo de execução. Além disto, os testes foram realizados em uma única arquitetura dentro das várias suportadas por Android, a arquitetura ARM (*big.little*). Para execução dos testes foram ligados o Wi-Fi e o Bluetooth, além de deixar rodando três aplicações (não especificadas) durante o tempo de execução em paralelo com o *benchmark* AnTuTu, tentando simular uma situação real.

4. Energy Assessment: o trabalho chamado “An Energy Assessment of Dalvik vs. ART” (Georgiev, Sillitti e Succi 2014) tem como objetivo comparar as duas máquinas virtuais do Android em relação ao consumo de energia, mostrando a diferença entre energia gasta e porcentagem de bateria descarregada. Para realizar esta análise, os autores executaram uma série de *benchmarks* a partir de uma configuração experimental idêntica em ambas as máquinas. Os *benchmarks* usados foram obtidos no Google Play, visando obter *benchmarks* que utilizassem diferentes componentes do dispositivo e tivessem um impacto relevante na execução para deixar a análise mais próxima à experiência que o usuário poderia ter ao utilizar um dispositivo. Além dos *benchmarks* utilizados para simular execução, foi utilizada uma ferramenta de medição da dissipação de potência (versão personalizada do PowerTutor). Cada teste foi reproduzido 30 vezes. Os testes foram realizados numa única arquitetura, ARM, e também considerando as tarefas em execução em segundo plano.

Nos últimos 3 trabalhos apresentados anteriormente, vemos que para a comparação feita pelos autores, foi utilizada somente uma arquitetura dentre as várias suportadas pelo Android (ARM). O mesmo foi feito neste trabalho. Além disto, dados das tarefas sendo executadas em segundo plano fazem parte dos dados coletados e com os quais fizeram a análise. Em contrapartida, no nosso trabalho fizemos uma análise de cada aplicação separadamente, sem levar em consideração os dados das tarefas executadas em segundo plano, isto foi feito para termos uma análise mais pontual sobre cada aplicação. Porém, cabe ressaltar que o nosso *profiler* permite a análise de todos os processos rodando ao mesmo tempo no SO. Outro ponto que diferencia a nossa proposta é que os trabalhos de Konradsson (2015) e Yadav e Bhadoria (2015) fizeram a análise comparativa baseada no desempenho,

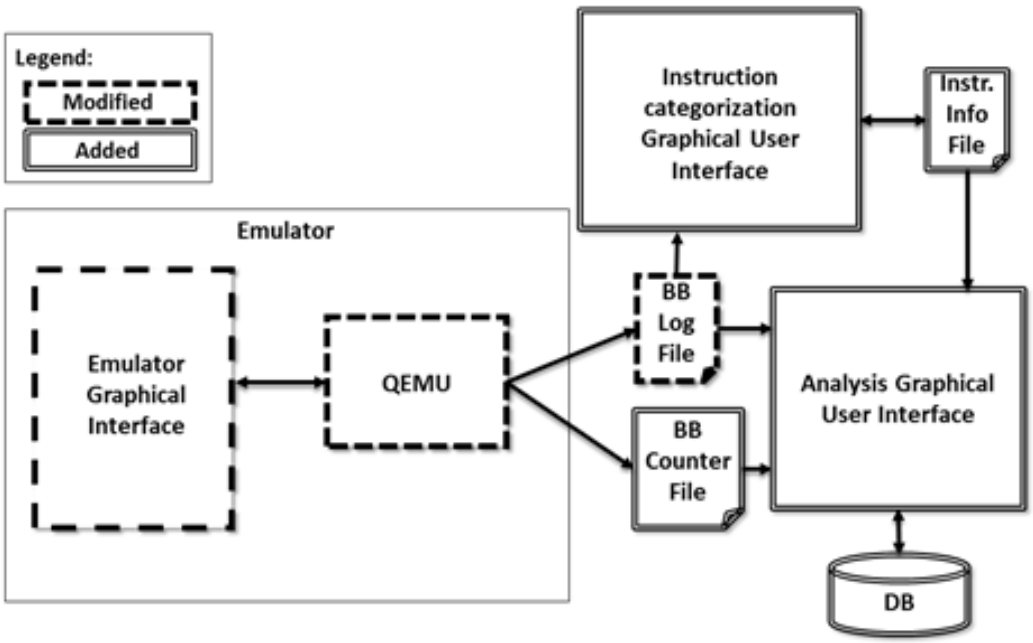
enquanto o (Georgiev, Sillitti e Succi 2014) fez baseado no consumo de energia. Porém, o nosso *profiler* é capaz de facilitar a análise comparativa em ambos os eixos.

4 ANDROPROF PARA ART

Como falado anteriormente, Android é organizado em diferentes camadas, sendo que as máquinas virtuais se encontram na camada das bibliotecas. O objetivo do trabalho apresentado é de fazer uma análise comparativa entre ambas as máquinas virtuais. Para isto realizamos uma extensão do *profiler* AndroProf (Sartor, Correa e Beck 2013) para suportar a máquina virtual ART.

O objetivo deste capítulo é de explicarmos o trabalho realizado para obtermos o AndroProf para ART. Para isso, começaremos falando da configuração do ambiente na primeira subseção, uma vez que é importante a familiarização com as ferramentas utilizadas para o desenvolvimento deste trabalho. É indispensável entender com mais detalhes o que é e como funciona o QEMU, assim como as modificações que foram feitas nele para podermos obter o comportamento esperado. Isto será explicado em detalhes na segunda subseção. Para finalizar este capítulo, falaremos sobre as interfaces gráficas (GUI) utilizadas como auxílio para análise dos dados coletados.

Figura 4.1 – Funcionamento do AndroProf para Dalvik



Fonte: Sartor, Correa e Beck (2013, p. 4)

Uma visão do funcionamento geral do AndroProf para Dalvik pode ser encontrada na Figura 4.1, onde vemos que foram modificados os elementos principais do QEMU, e junto a eles foram utilizadas as duas interfaces de usuário desenvolvidas por Sartor, Correa e Beck

(2013), as quais processam os dados gerados pelo emulador e fazem uso de um banco de dados.

4.1 Ambiente de trabalho

Para o desenvolvimento do nosso trabalho foi obtido, do site do Android Source (2016), o SDK das versões desejadas, versões 4.0 e 4.4 conforme a escolha de versionamento apresentada na subseção 2.6.

O emulador Android utiliza o *Android Virtual Device* (Android AVD 2016), que faz parte do SDK, para identificar a configuração do dispositivo que será emulado. Através do AVD as configurações de software e hardware a serem emuladas são definidas, permitindo assim modelar qualquer dispositivo suportado.

O SDK do Android provê o Gerenciador de SDK Android, o qual possibilita o gerenciamento de versões Android. Neste gerenciador são encontradas as imagens das arquiteturas fornecidas para alguma versão específica deste SO. Estas imagens são posteriormente utilizadas para a criação da AVD que será emulada. Para o funcionamento do emulador é preciso obter pelo menos uma arquitetura. Porém, notamos que na versão 5.0 não era disponibilizada nenhuma, enquanto que para a versão 4.4 encontramos duas das três arquiteturas para as quais foi projetado inicialmente nosso *profiler*: ARM e x86 (não fornecendo a MIPS). Foram então obtidas estas arquiteturas a partir do Gerenciador de SDK Android.

Após ter ambos SDKs e termos as imagens das arquiteturas que possibilitassem a extensão do *profiler*, foram feitas as modificações dentro do QEMU nas versões do SDK obtidas. Estas modificações serão apresentadas na próxima subseção.

O Android ABD (2016) é uma ferramenta versátil disponibilizada pelo Android no seu SDK, que funciona tanto para emuladores quanto para dispositivos conectados, e utilizado para instalar uma série de *benchmarks* no dispositivo, assim como para acessar e gerenciar o dispositivo. Os *benchmarks* que foram instalados serão apresentados no capítulo 5.

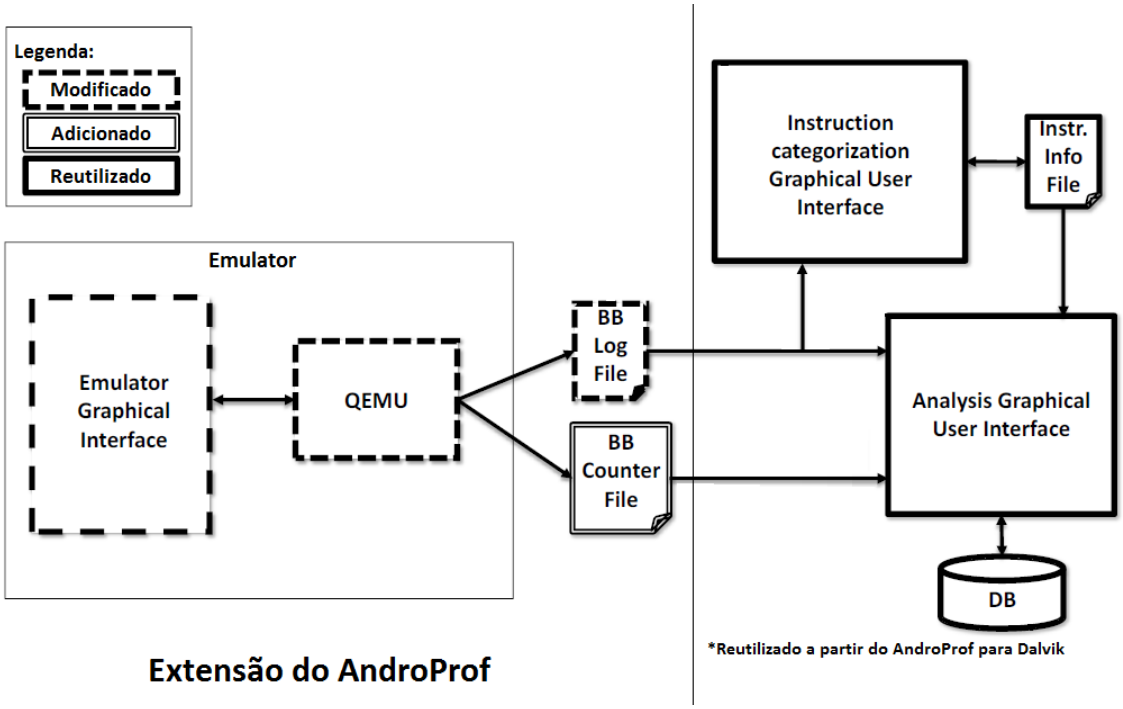
Depois de termos instalado os *benchmarks* nas AVDs criadas especificamente para cada versão, eles foram executados para coletar dados, que foram posteriormente analisados. Os resultados desta análise serão apresentados no capítulo 6, com o fim de apresentarmos o impacto da máquina virtual ART em relação à Dalvik.

4.2 QEMU

Como dito anteriormente, o QEMU é um emulador utilizado como base para emular um processador rodando o Android, e emula uma plataforma virtual de hardware para diferentes arquiteturas, através do seu mecanismo de tradução binária. Além disto, é o emulador oficial e mais usado para desenvolver e testar aplicações Android.

Dentro de cada SDK de cada versão do Android é encontrada uma cópia adaptada/modificada do QEMU para aquela respectiva versão deste SO. Por isto, conforme dito na subseção 2.6, fez parte do trabalho uma análise dos códigos do QEMU nas versões recentes do Android, pois este código foi alterado em função da introdução da nova máquina virtual. Para estender o AndroProf precisamos obter o SDK da versão 4.4, que por padrão contém Dalvik, mas que para desenvolvedores permite a utilização da ART.

Figura 4.2 – Funcionamento do AndroProf para ART



Fonte: o autor

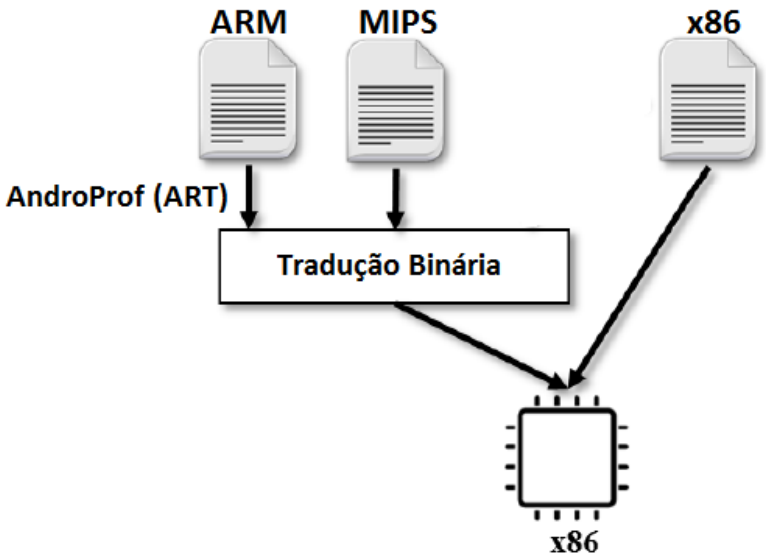
Devido a cada versão do SDK possuir seu próprio QEMU modificado embutido, a extensão do *profiler* deve ser feita para cada versão a ser utilizada, no nosso caso, essa extensão foi feita na versão 4.4. Podemos ver na Figura 4.2, que os elementos à direita, são

componentes reutilizado da versão original do AndroProf, enquanto que os componentes à esquerda tiveram de ser modificados nesta nova versão do nosso *profiler*.

As modificações apresentadas nesta subseção são referentes à Figura 4.2, mais especificamente ao lado esquerdo da figura, o qual contém o QEMU.

Pela máquina hospedeira basear-se na arquitetura x86, o processador emulado x86 pode ser rodado diretamente no processador, enquanto que as arquiteturas ARM e MIPS necessitam passar pelo processo de tradução binária. Porém, a informação que o *profiler* coleta é extraída no processo de tradução. Neste trabalho o *profiler* foi adaptado unicamente para a ARM, como podemos ver na Figura 4.3.

Figura 4.3 – Tradução binária, extensão do AndroProf para suportar ART

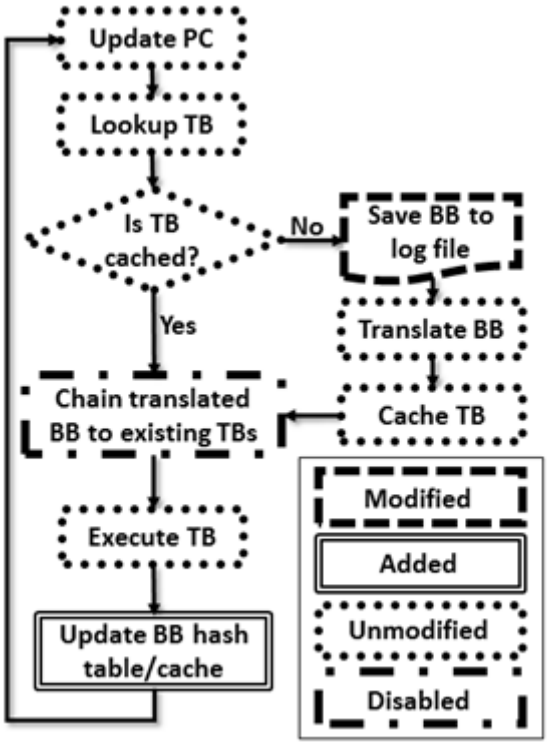


Fonte: o autor

Como já mencionado, os dados que são coletados são no nível das instruções, e para isto o QEMU foi modificado. O QEMU disponibiliza um mecanismo de log dos blocos básicos ao qual foi adicionada a identificação dos BBs. Estes blocos básicos são salvos no arquivo 'BB Log File', da Figura 4.2, que posteriormente é utilizado pelas GUIs. Com isto é possível analisar cada bloco básico e as suas instruções.

Através dos PIDs (identificação do processo), é também identificada cada aplicação. Para ter controle do uso dos processos e dos BBs é utilizada uma tabela hash, responsável por salvar quantas vezes foi executado cada BB e por qual processo. Para obter este resultado, o fluxo do QEMU teve que ser modificado (Figura 4.4) da mesma forma que foi modificado no trabalho original do AndroProf.

Figura 4.4 – Fluxo do QEMU modificado



Fonte: Sartor, Correa e Beck (2013, p. 4)

O fluxo modificado do QEMU começa verificando, cada vez que se tem um bloco básico a ser executado, se aquele bloco está salvo em cache (bloco já traduzido). Se o bloco ainda não foi traduzido, então o QEMU traduz para um Bloco Traduzido (BT). Por outro lado, se o bloco foi traduzido então o seu BT correspondente é carregado da cache, sem precisar traduzir o BB de novo. BT é um bloco básico composto por instruções suportadas pela arquitetura hospedeira (x86).

Para agilizar o processo de simulação, foi inserida uma cache de software para a memoria cache, com mapeamento direto e 2048 slots. Esta cache obteve hit ratio de 97% na versão para Dalvik. Portanto, também foi utilizada para a ART, obtendo hit ratio de 94,4%.

Também foi inserido um mecanismo de seleção de rastreamento, que permite habilitar ou desabilitar a tabela hash e sua memória cache, possibilitando ao desenvolvedor escolher os momentos em que fará o rastreamento das suas aplicações. Ao fechar a GUI do emulador, todas as informações obtidas são salvas no arquivo ‘BB Counter File’ da Figura 4.2, para posterior análise, que pode ser feita pelas GUIs que serão apresentadas na próxima seção.

Este é o funcionamento do AndroProf para ambas as máquinas. A única diferença entre elas é que a versão que suporta Dalvik é multi-arquitetural, enquanto que a versão para ART suporta somente ARM, devido à indisponibilidade da arquitetura MIPS para teste e a não termos mudado o fluxo do comportamento da arquitetura x86.

Para estendermos o *profiler*, foram analisadas as novas versões do QEMU, e concluímos que as modificações poderiam ter sido feitas até a versão 5. Porém, devido à compilação, não teríamos como testar o comportamento nessa versão. Dessa forma, as modificações que já tinham sido feitas na versão que suporta Dalvik foram necessárias também na versão 4.4.

Resumindo as modificações feitas no código fonte do QEMU do novo SDK: colocamos um identificador para cada bloco básico de código, identificamos as aplicações através do PID do seu processo, modificamos o fluxo do QEMU, conforme a Figura 4.4, para utilizar a tabela hash (adicionada), e inserimos o mecanismo de seleção de rastreamento.

4.3 Interfaces

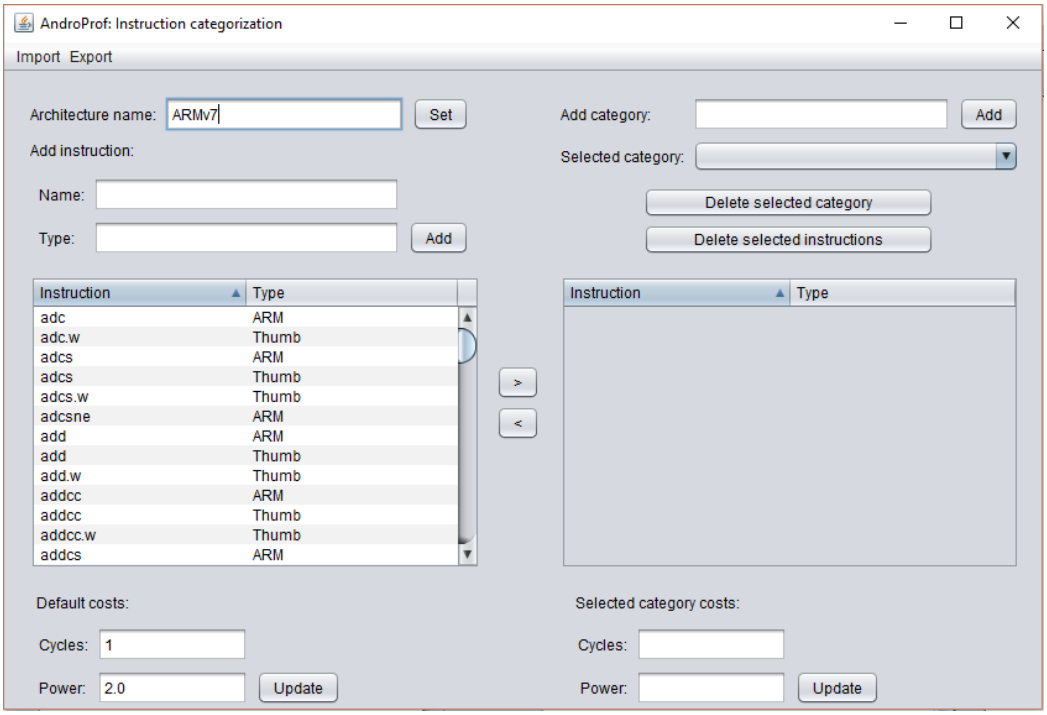
Nas Figuras 4.1 e 4.2 vemos que existem componentes que foram adicionados para fazer parte do *profiler*, e outros componentes que foram modificados para o funcionamento do mesmo. Os componentes que foram adicionados por Sartor, Correa e Beck (2013) são reutilizados na versão que suporta ART (lado direito da Figura 4.2). Para poder reutilizar estes componentes foi modificado o QEMU a modo de obtermos o mesmo formato de entrada para estes componentes, o mesmo formato de arquivo. A seguir serão descritas as GUIs que foram adicionadas no *profiler* original que suporta Dalvik e que reutilizamos na extensão.

4.3.1 Instruction Categorization GUI Tool

A Instruction Categorization GUI nos permite classificar em categorias as instruções executadas pelo emulador, a partir de um arquivo de log de instruções criado pelo QEMU modificado. Como saída, esta GUI gera um arquivo XML que contém informações sobre as instruções (o arquivo Instr. Info File da Figura 4.2), tais como média de número de ciclos e dissipação de potência para cada categoria disponível. Estas informações contidas no arquivo de saída podem ser customizadas pelo desenvolvedor na GUI.

Esta GUI nos permite classificar todas as instruções de um dado tipo para uma dada categoria em um único passo, assim como permite reuso de arquivos de categorização, através da importação dos mesmos.

Figura 4.5 – AndroProf Instruction Categorization GUI



Fonte: Sartor, Correa e Beck (2013, p. 4)

Como podemos ver na Figura 4.5, esta GUI contém duas tabelas, a do lado esquerdo contém as instruções que não fazer parte de nenhuma categoria, as quais a ferramenta classifica como “indefinida” e possui custo padrão (inserido na parte inferior à esquerda). A tabela do lado direito contém as instruções de uma dada categoria, a qual pode ser selecionada no combo-box Selected category, localizado acima desta tabela, e à qual podem ser atribuídos custos diferentes aos atribuídos para as instruções sem categoria (na parte inferior à direita).

Esta ferramenta permite aos desenvolvedores e pesquisadores classificar qualquer tipo de instrução independente da ISA (Arquitetura do Conjunto de Instruções) à qual pertence e do seu tamanho de codificação. Vemos na Figura 4.5 que aparecem dois tipos de instruções, Thumb e ARM, a primeira faz parte do da ISA da ARM, porém com tamanho de codificação menor (16 bits) que a segunda (32 bits). Por exemplo, podemos considerar a arquitetura ARMv7, que contém uma categoria de branch condicional de 3 ciclos (Infocenter) e dissipação de potência de 113miliWatts para cada instrução, conforme Bazzaz, Salehi e Ejlali (2013). Estas características permitem a criação da categorização das instruções BEQ (ARM),

BEQ (Thumb), BNE (ARM), BLT (Thumb) assim como outras instruções de branch condicional.

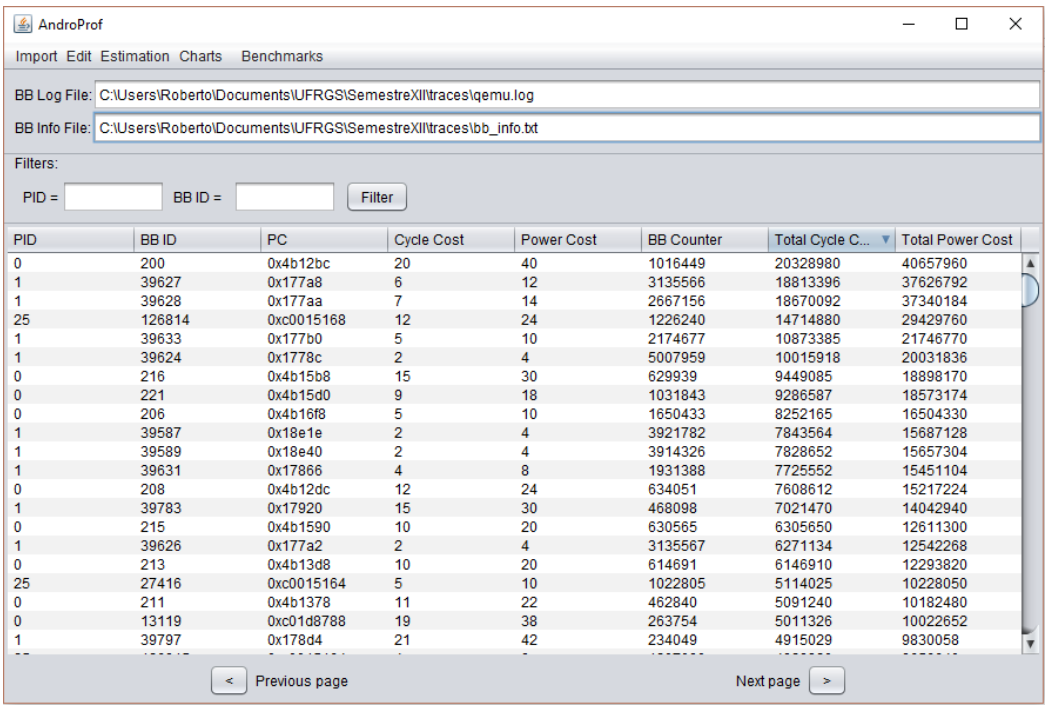
4.3.2 Analysis GUI Tool

A segunda GUI, Analysis GUI, importa os dois arquivos gerados pelo QEMU mais o arquivo gerado pela Instruction Categorization GUI. Depois de importar os 3 arquivos, processa e apresenta os dados coletados, de uma forma que facilita o entendimento dos dados, e salva eles num banco de dados.

Esta ferramenta nos permite obter informações como consumo de energia e número de ciclos de cada processo em forma de gráfico, também nos permite obter informações por PID dos processos em histogramas, nos mostrando assim quais são os processos mais custosos. Além disso, nos mostra o consumo de energia e o desempenho para alguma frequência dada. Permite importar categorizações de instruções para diferentes conjuntos de arquiteturas ou organizações. Possibilita a avaliação de processos separadamente.

Em resumo, coleta informações sobre os 3 arquivos de entrada e permite vários tipos de análise sobre os dados coletados, visando a apresentação destes de forma que facilite o entendimento do usuário. A janela principal desta GUI está representada na Figura 4.6.

Figura 4.6 – AndroProf Analysis GUI



Fonte: Sartor, Correa e Beck (2013, p. 5)

5 BENCHMARKS E AVALIAÇÃO DA MÁQUINA VIRTUAL ART

Depois de termos modificado o código fonte do QEMU, obtendo o comportamento desejado no Android, para analisarmos o impacto da máquina virtual ART em relação à Dalvik, foi utilizado um conjunto de *benchmarks*, os quais são apresentados na subseção 5.1. Estes *benchmarks* visam provocar o uso dos diferentes componentes do processador. Na segunda subseção abordaremos a avaliação da máquina virtual ART, a qual visa obter um resultado completo sobre o impacto desta máquina virtual em sistemas Android.

5.1 Benchmarks

Para nossa análise comparativa entre ambas as máquinas virtuais foi utilizado um conjunto de 5 *benchmarks*, o qual é um subconjunto dos *benchmarks* utilizados por Sartor, Correa e Beck (2013), obtidos da SPEC, originalmente desenvolvidos para análise de desempenho do JRE (*Java Runtime Environment*).

Como falado anteriormente, Android pode ser executado tanto com código Java quanto com código nativo (JNI). Os *benchmarks* foram analisados no trabalho do AndroProf para Dalvik pelos autores com ajuda do *profiler*, de modo a encontrar os métodos responsáveis pelo maior tempo de uso da aplicação. Desta maneira, os métodos que executavam mais de 10% do tempo total da aplicação foram traduzidos para código nativo, os quais podem ser chamados pela JNI, sem precisarem passar pela máquina virtual, como descrito na subseção 4.2. Sete dos *benchmarks* utilizados pelos autores de AndroProf para Dalvik possuem pelo menos um método que alcança esta taxa.

Tendo os métodos mais utilizados escritos em código nativo, eles organizaram os *benchmarks* em dois conjuntos: o primeiro é o conjunto de 12 *benchmarks* escritos em Java obtidos da SPEC, e o segundo é um conjunto de 7 *benchmarks* com partes de código escritos em Java, com seu respectivos métodos mais usados escritos em código nativo. Porém, por limitações da máquina encarregada de rodar o emulador, não foi possível executar todas as aplicações dos *benchmarks*. Dessa forma, para este trabalho foram usados 5 *benchmarks* do primeiro conjunto e 2 do segundo conjunto, conforme vemos na Tabela 5.1.

Tabela 5.1 – Conjuntos de Benchmarks

Aplicações Java	*Aplicações JNI*
Scimark FFT	Scimark FFT
Scimark SOR	Scimark SOR
Serial	
Crypto RSA	
SQLite	

Fonte: o autor

Analisando o código dos *benchmarks* vimos que existe uma diferença entre os *benchmarks* Scimark FFT e o Scimark SOR em relação ao resto. Os dois citados anteriormente possuem mais atribuições, cálculos matemáticos, declarações e laços, enquanto que os outros possuem mais troca de informações, como passagem de parâmetros e chamadas de funções.

Estes conjuntos foram utilizados na avaliação da máquina virtual ART, escolhidos para obtermos quantitativamente o impacto da máquina virtual ART. Esta avaliação será apresentada na próxima subseção.

5.2 Avaliação da Máquina Virtual ART

Para avaliar a máquina virtual ART, com a ajuda do nosso *profiler*, dividimos nossa avaliação em duas partes, as quais serão apresentadas nas seguintes subseções.

5.2.1 ART vs Dalvik executando com código Java

Como falado na subseção 5.1, os *benchmarks* obtidos foram organizados em dois conjuntos. Para avaliarmos o impacto da máquina virtual ART em relação à Dalvik foi utilizado o conjunto que contém os 5 *benchmarks* escritos em código Java, uma vez que, como explicado na subseção 2.2, a linguagem de programação Java precisa ser interpretada pela máquina virtual para poder ser executada na arquitetura alvo, enquanto que o código nativo não precisa ser interpretado pela máquina virtual.

Por este conjunto possuir todo seu código escrito em Java, o qual deve passar pela máquina virtual, ele nos proporciona um resultado que leva em consideração o uso máximo da máquina virtual sendo utilizada durante a execução destas aplicações, o que espelha de

uma forma concreta o impacto da máquina virtual ART em relação à Dalvik. Assim, esta avaliação visa explorar o uso das máquinas virtuais.

5.2.2 Java vs JNI na máquina virtual ART

Para obtermos uma análise mais profunda do impacto da máquina virtual ART em aplicações Android, esta VM também será analisada em relação à linguagem em que o código das aplicações foi desenvolvido.

O segundo método de análise usa o segundo conjunto de *benchmarks*, o qual contém partes de código em Java assim como partes em código nativo (métodos mais utilizados). A execução de aplicações com ambas as linguagens é possível através da JNI, a qual possibilita a interação entre código Java e código nativo. Esta avaliação visa observar o impacto da máquina virtual ART quando executados os métodos mais utilizados em Java em relação à execução dos mesmos em código nativo. Lembrando que a máquina virtual ART utiliza o modo de compilação AOT, o qual interpreta os *bytecodes* Java, enquanto que o código nativo só precisa ser compilado. Visamos assim, avaliar o impacto desta interpretação dos *bytecodes* em relação à compilação do código nativo.

6 RESULTADOS

Depois de termos executado os conjuntos de *benchmarks* no emulador Android e com a ajuda do nosso *profiler*, foram obtidos os resultados da nossa análise, uma vez que o *profiler* coleta dados como o número de instruções e o número de blocos básicos executados por cada aplicação, os quais são identificados pelo seu PID. Além disso, também permite obter informações sobre desempenho e consumo de energia.

Nas seguintes subseções, apresentaremos os resultados obtidos através da execução dos dois métodos de análise apresentados anteriormente. A primeira subseção mostrará o impacto da máquina virtual ART em relação à Dalvik, enquanto que a segunda subseção discutirá o impacto das aplicações Java em relação às aplicações mistas (Java e JNI) executadas em ART.

6.1 ART vs Dalvik

Para obtermos os resultados deste método (apresentado na subseção 5.2.1), foram analisados os resultados obtidos por Sartor, Correa e Beck (2013) para a máquina virtual Dalvik, na arquitetura ARM, em relação ao conjunto de *benchmarks* que contém somente código Java. Então, foi utilizada a extensão do *profiler*, a qual suporta ART, para rodar o mesmo conjunto de *benchmarks* na mesma arquitetura falada anteriormente. Após a execução destes *benchmarks*, os dados coletados foram analisados e comparados, os resultados serão apresentados a seguir.

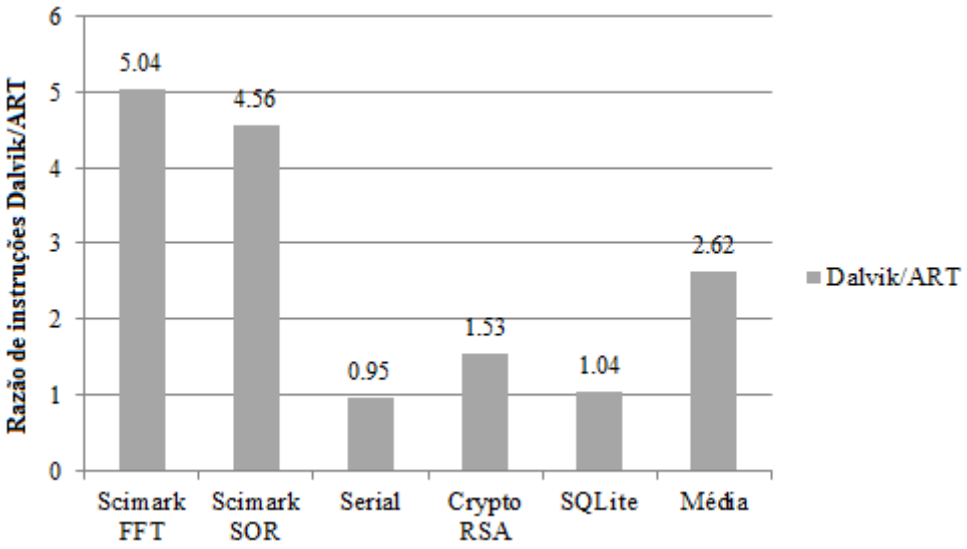
6.1.1 Instruções executadas

Quando apresentadas ambas as máquinas virtuais, no capítulo 2, falamos sobre o modo de compilação de cada uma delas. O modo de compilação influencia diretamente o número de instruções executadas por uma aplicação. Compilação JIT, utilizada pela Dalvik, interpreta os *bytecodes* para instruções da arquitetura alvo em tempo de execução. Em contrapartida, a compilação AOT utilizada pela ART, interpreta esses *bytecodes* durante a instalação da aplicação. Com isto, otimizações no código, que seriam muito custosas durante o tempo de execução, podem ser feitas na instalação.

Estas otimizações são refletidas no número total de instruções executadas. Porém, em alguns casos não é viável fazer estas otimizações, uma vez que o código pode ser altamente

dependente do fluxo ou dos dados, ou ter muita comunicação entre objetos e muita troca de contexto. Com a ajuda do nosso *profiler*, foi possível obter o número de instruções executadas por cada uma das máquinas rodando a mesma aplicação (mesmo código) para ambas as máquinas. Podemos observar, pela Figura 6.1, que o número de instruções necessárias na Dalvik para executar uma aplicação é maior que o número que ART precisa. Esta figura leva em consideração a razão de instruções executadas pela Dalvik em relação à ART. Como podemos ver, o número de instruções executadas pela Dalvik é 2,62 vezes maior, em média, em relação a ART.

Figura 6.1 – Razão de instruções Dalvik/ART



Fonte: o autor

Vemos na Tabela 5.1 que os *benchmarks* Scimark FFT e Scimark SOR estão em ambos os conjuntos, o que quer dizer que ambos *benchmarks* têm pelo menos um método que executa durante mais de 10% do tempo da aplicação. Pela análise feita anteriormente sobre os *benchmarks* concluímos que maior número de otimizações é possível de serem aplicadas a estes, refletindo no número total de instruções e na razão de instruções Dalvik/ART, como podemos observar na Figura 6.1. Otimizações tais como substituição/eliminação de expressões redundantes ou desnecessárias, desmembramento de laços ou até substituição de operações, como, por exemplo, shift no lugar de multiplicações. Por estas características, estes dois *benchmarks* apresentam uma razão maior que a do resto de *benchmarks* do conjunto. Podemos ver que somente em um dos casos o número de instruções foi menor quando executado na Dalvik. No *benchmark* Serial, o qual trabalha com muita comunicação entre métodos, otimizações dificilmente podem ser aplicadas, ainda assim o número de

instruções executadas por ambas as máquinas virtuais é praticamente o mesmo (razão de 0,95).

6.1.2 Desempenho e Consumo de Energia

Para estimarmos desempenho e consumo de energia foram utilizadas as mesmas informações do trabalho feito por Sartor, Correa e Beck, apresentadas a seguir (Tabela 6.1), uma vez que nossa comparação neste trabalho leva em conta o mesmo tipo de configuração de dispositivo assim como os dados coletados por eles. Por fazermos uso das mesmas organizações e arquiteturas que eles utilizaram, nossos dados também podem ser analisados com estas informações.

As informações referentes às organizações ARM, Cortex-A8 e Cortex-A9, são apresentadas na Tabela 6.1, obtidas dos artigos de Blem, Menon e Sankaralingam (2013).

Tabela 6.1 – Informações de potência e de ciclos para ARM

	Cortex-A8	*Cortex-A9*
Frequência (GHz)	0,6	1,0
Dissipação de Potência do Processador (W)	0,85	1,2
CPI	3,36	2,17
Taxa de acesso a Memória Principal	~1%	~1%

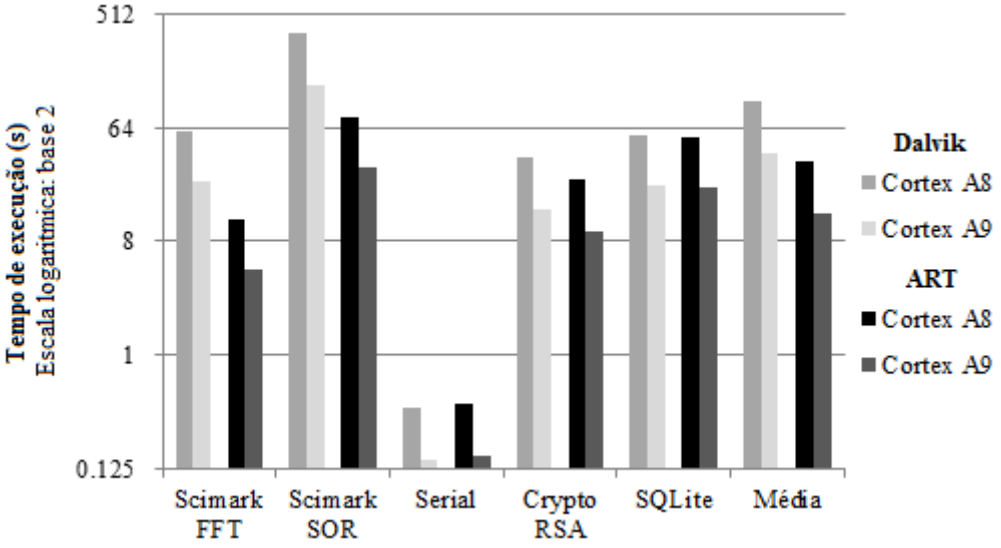
Fonte: Blem, Menon e Sankaralingam, K 2013

Para os cálculos também serão utilizadas as informações sobre dissipação de potência para leitura e escrita da memória principal obtidas do CACTI 6.5 pelos autores do AndroProf para Dalvik, onde a configuração de memória é 512MB, 8 bancos de memória, tamanhos de blocos de 64 bytes e tecnologia de 45nm. O resultado dessa configuração é de 8,26ns para acesso a memória com consumo de energia de 2,66nJ/2,56nJ para leitura e escrita, respectivamente, e potência estática de 109,613mW. Além disso, assim como no trabalho de Sartor, Correa e Beck, será utilizada a média do consumo de energia de leitura e escrita em memória, devido à pequena diferença entre elas, dessa forma um consumo de 2,61nJ será utilizado para o cálculo de consumo de energia, independentemente se é leitura ou escrita a memória. Por não existir um simulador de memória cache no QEMU é preciso utilizar estas informações para estimarmos desempenho e consumo de energia.

A Figura 6.2 nos mostra o tempo estimado de execução para cada aplicação em cada uma das organizações das quais obtivemos os dados. Vemos que, em média, a organização A8 consegue um tempo de execução 2,58x maior que o tempo de execução da A9, e o tempo

de execução mantém a mesma proporção que a respectiva para número de instruções (2,62), uma vez que o CPI e a frequência destas organizações não mudou. Podemos ver também, que o *benchmark* Serial é o único que apresenta um desempenho relativamente melhor quando executado na Dalvik, devido ao número de instruções, como comentado na subseção anterior.

Figura 6.2 – Tempo estimado de execução

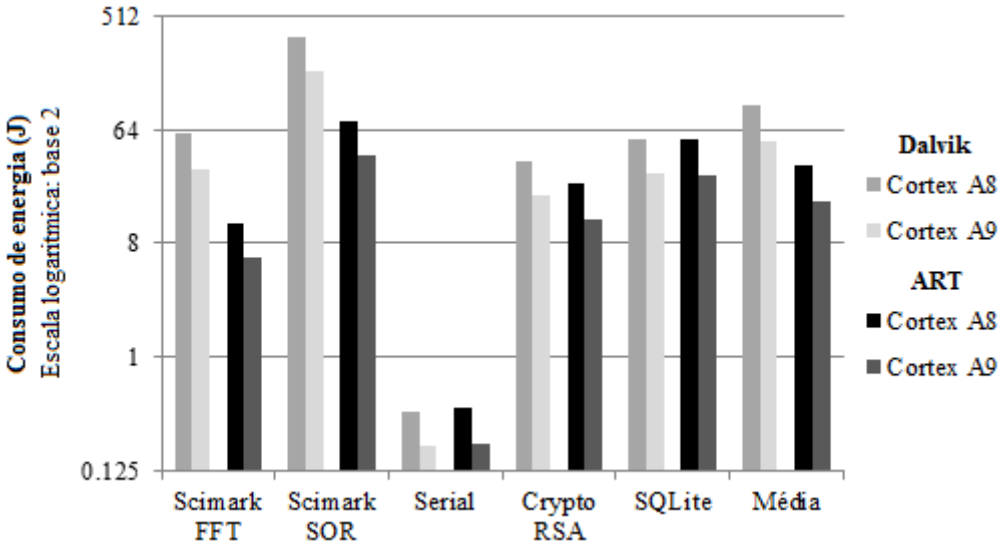


Fonte: o autor

Para terminar a nossa análise comparativa entre máquinas virtuais, a Figura 6.3 mostra o consumo de energia para cada aplicação em cada uma das organizações, do mesmo modo que foi mostrado o desempenho. Para obtermos estes resultados em relação à energia, foram utilizadas as informações sobre acesso a memória antes apresentadas, assim como o tempo de execução de cada aplicação. Notamos que, em termos de consumo de energia, a organização A8 gasta 1,88x mais que a A9. Quando levando em consideração as máquinas virtuais o consumo de energia se mantém proporcional ao tempo de execução de cada um.

Após obtermos nossos resultados sobre esta análise, foi feita uma comparação deles em relação aos resultados obtidos pelos autores dos trabalhos relacionados. Mesmo alguns autores não apresentando as características das aplicações rodadas, os resultados obtidos em todos os trabalhos são muito similares. Vemos que em média, a ART apresenta desempenho e consumo de energia mais eficiente que a Dalvik, independente de os resultados levarem em consideração as tarefas sendo executadas em segundo plano ou não.

Figura 6.3 – Consumo de energia



Fonte: o autor

6.2 Java vs JNI em ART

Nesta subseção é feita uma análise similar à análise feita no método anterior, a diferença neste método de análise é que a comparação não está sendo feita entre máquinas virtuais, mas entre aplicações com diferentes tipos de código: aplicações com código puramente Java e aplicações com código misto, que contém partes de código Java e partes de código nativo.

Para facilitarmos a leitura, nesta subseção chamaremos as aplicações que contém somente código Java como aplicações Java, enquanto que as aplicações que contém tanto partes de código em Java quanto em código nativo serão chamadas de aplicações JNI. Outro ponto a ser levado em consideração é que ambos os tipos de aplicação foram executados na máquina virtual ART, visando obtermos o impacto da máquina virtual ART quando executadas aplicações com mais uso da VM (aplicações Java) em relação a aplicações com menos uso desta mesma (aplicações JNI).

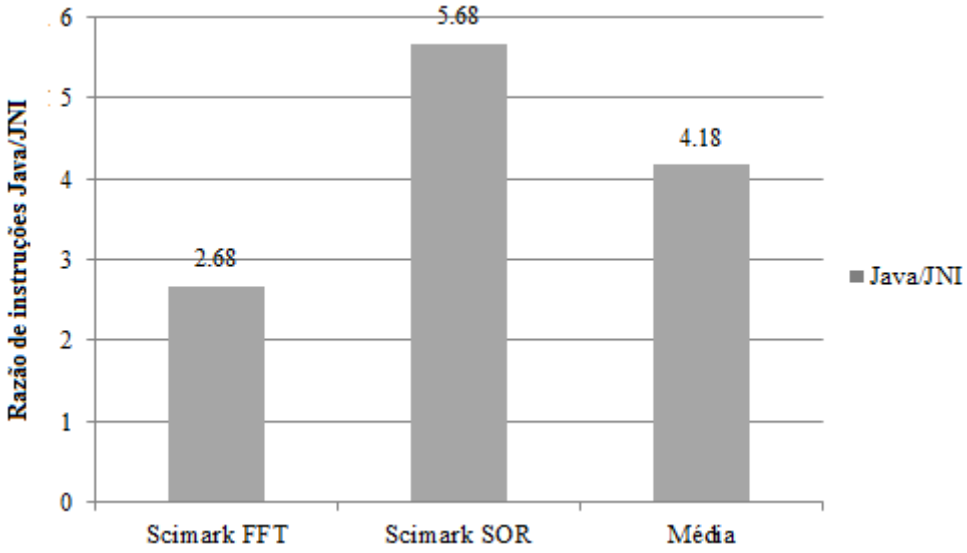
6.2.1 Instruções executadas

Do mesmo modo que obtido o número de instruções para a análise do método anterior, para este método utilizamos o *profiler* AndroProf para este fim. A Figura 6.4 nos mostra de forma clara a diferença das instruções executadas por cada tipo de aplicação, apresentando a

razão de instruções Java/JNI. Vemos que ambas as aplicações Java utilizam mais instruções para serem executadas do que as aplicações JNI. Quando esta razão é maior, significa que o impacto do uso da máquina virtual ART é maior, podemos concluir, pelas aplicações utilizadas, que a máquina virtual ART influencia mais o *benchmark* Scimark SOR em relação ao Scimark FFT.

Na análise feita por Sartor, Correa e Beck (2013), a qual analisou a VM Dalvik, vemos que estes mesmos *benchmarks* tiveram uma taxa de instruções Java/JNI de 5,34 e 7,12 para FFT e SOR respectivamente, para ARM. Podemos ver que a razão, em média, diminui quando executadas as aplicações na ART, o que significa que a alteração da máquina virtual também traz benefícios em relação a aplicações JNI, uma vez que a parte de código escrita em Java pode ser otimizado na instalação da aplicação, devido à compilação ser AOT.

Figura 6.4 – Razão de instruções Java/JNI

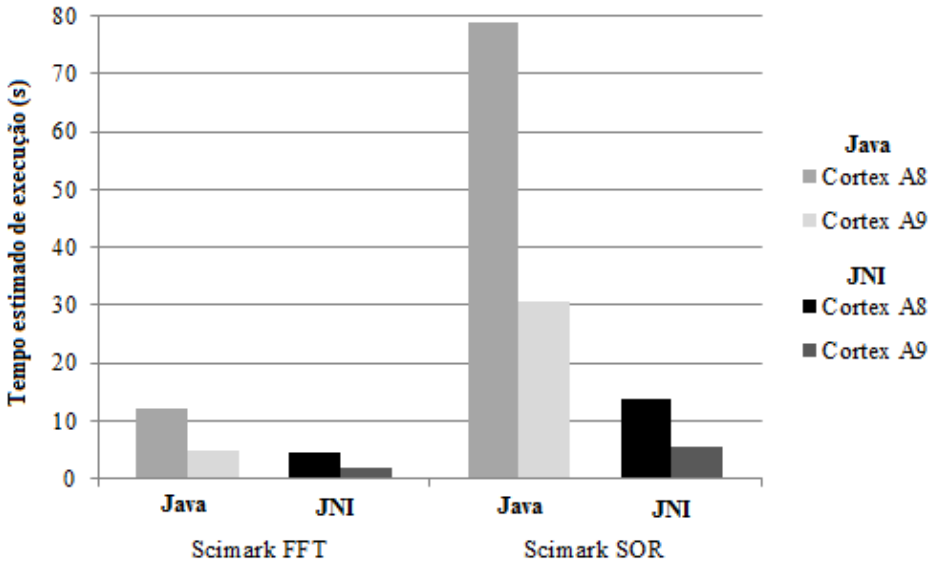


Fonte: o autor

6.2.2 Desempenho e Consumo de Energia

Devido ao número menor de instruções, vemos na Figura 6.5 que o desempenho das aplicações JNI é melhor, levando menos tempo de execução. Porém, a relação entre ambas as organizações se mantém igual, com um desempenho de 2,58x melhor para a organização A9, uma vez que o CPI e a frequência se mantêm a mesma.

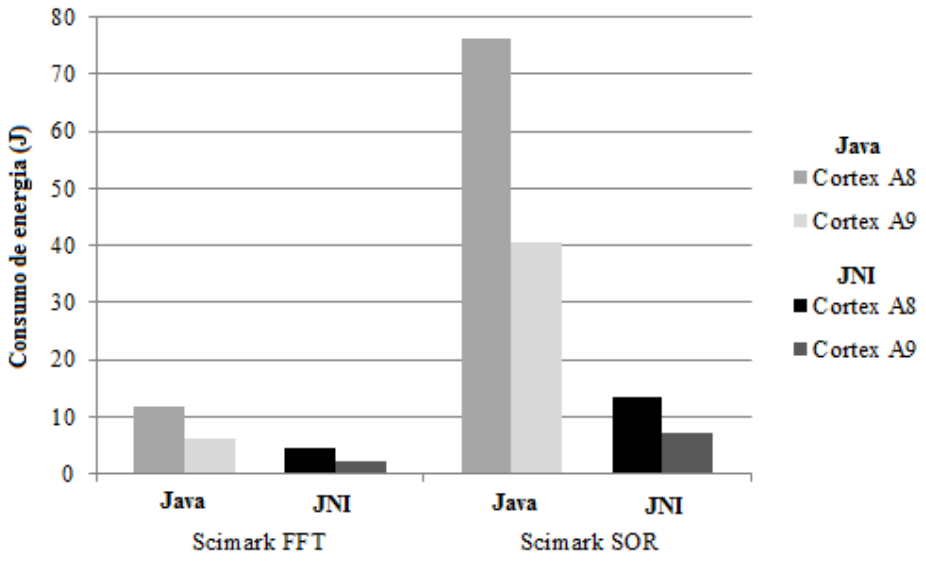
Figura 6.5 – Tempo estimado de execução



Fonte: o autor

Do mesmo jeito que o desempenho é influenciado devido ao número de instruções executadas em cada tipo de aplicação, o consumo de energia é também influenciado. Podemos ver na Figura 6.6 que as aplicações JNI, para os *benchmarks* executados, consomem menos energia que as aplicações Java. Porém, cabe ressaltar que, como mostrado no trabalho do AndroProf para Dalvik, isto não é o comportamento para todas as aplicações, uma vez que existem aplicações nas quais o uso de código nativo não amortece o custo da troca de contexto, a qual envolve cópia dos operandos em memória, de um lado para o outro (de Java para código nativo ou vice-versa).

Figura 6.6 – Consumo de energia



Fonte: o autor

Podemos concluir por estes resultados que a compilação pelo GCC, a partir do C, ainda é mais eficiente em relação à compilação feita pela ART para os *bytecodes* de entrada. Porém, vemos que em relação à Dalvik, a ART se aproximou aos resultados obtidos para o uso de código nativo e compilação pelo GCC.

.

7 CONCLUSÃO

Através dos resultados dos dois métodos de análise e dos conceitos teóricos envolvidos neste trabalho podemos concluir que a troca de máquina virtual da Dalvik para a ART, feita pela Google, é de fato uma mudança importante para sistemas Android, fazendo estes utilizarem um número menor de instruções, o que implica num menor consumo de energia e uma melhor experiência de uso, quesito importante para qualquer usuário deste SO. Com isto, vemos que o uso do modo de compilação AOT é completamente viável para uso em dispositivos móveis, uma vez que estes dispositivos cada vez mais estão incrementando em relação ao espaço de disco e memória, limitações importantes pelo antigo uso do modo de compilação JIT.

Vemos que esta mudança de máquina virtual traz benefícios, em média, para qualquer tipo aplicação, mas que beneficia mais as aplicações Java em relação às JNI. Também, a nova máquina reduziu a diferença entre a compilação de código nativo e o código Java. Para finalizar sabemos que ART foi projetado para ser mais eficiente em desempenho e consumo de bateria, através dos resultados obtidos, vemos que o objetivo inicial da Google foi alcançado.

Como trabalhos futuros podemos fazer o AndroProf para ART suportar as arquiteturas MIPS e x86 também, para assim cobrirmos as nove características desejadas para nosso *profiler*. Também, no lugar de utilizar os dados obtidos para estimar o consumo de energia, pode ser feito um simulador de acesso à memória cache capaz de simular vários modelos, e importar ele ao AndroProf.

Para estimar desempenho e consumo de energia de uma forma mais exata, podemos criar categorias de instruções com seus respectivos custos de ciclos e potência, uma vez que nosso *profiler* está pronto para usar este tipo de categorização, mas neste trabalho não foi utilizada esta ferramenta por não termos conseguimos informações para uma dada categoria de instruções.

REFERÊNCIAS

ARM. **Inscructions cycle count.** Disponível em <http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ddi0214b/CHDGHAAG.html>. Acesso em: 22 nov. 2016.

BAZZAZ, M., SALEHI, M., EJLALI, A. **An accurate instruction-level energy estimation model and tool for embedded systems**. IEEE Transactions on Instrumentation and Measurement, v. 62, n. 7, p. 1927-1934, 2013.

BLEM, E., MENON, J., SANKARALINGAM, K. **A detailed analysis of contemporary arm and x86 architectures**. UW-Madison Technical Report. 2013.

BLEM, E., MENON, J., SANKARALINGAM, K. **Power struggles: Revisiting the RISC vs. CISC debate on contemporary ARM and x86 architectures.** High Performance Computer Architecture (HPCA2013), 2013 IEEE 19th International Symposium on (pp. 1-12). IEEE. 2013.

COUTO, M., CARÇAO, T., CUNHA, J., FERNANDES, J. P. E SARAIVA, J. **Detecting anomalous energy consumption in android applications**. Programming Languages (pp. 77-91): Springer International Publishing, 2014.

DEVELOPER ANDROID. **Android ABD**. Disponível em <https://developer.android.com/studio/command-line/adb.html>. Acesso em: 03 nov. 2016.

DEVELOPER ANDROID. **Android AVD**. Disponível em <http://developer.android.com/tools/devices/index.html>. Acesso em: 20 set. 2016.

DEVELOPER ANDROID. **Android Device Monitor**. Disponível em <https://developer.android.com/studio/profile/monitor.html>. Acesso em: 20 set. 2016.

DICKMAN, L.., INDIAN ROCKS BEACH, F. L. **A Comparison of Interpreted Java, WAT, AOT, JIT, and DAC**. 2002.

GEORGIEV, A. B., SILLITTI, A. E SUCCI, G.. **Open Source Mobile Virtual Machines: An Energy Assessment of Dalvik vs. ART**. OSS (pp. 93-102), 2014.

GOOGLE PLAY. **PowerTutor**. Disponível em <https://play.google.com/store/apps/details?id=edu.umich.PowerTutor&hl=pt_BR>. Acesso em: 20 set. 2002.

KAELI, D., & SACHS, K.. Teste e projeto visando o teste de circuitos e sistemas integrados. In: REIS, R. A. da L. (Ed.). **Computer Performance Evaluation and Benchmarking: SPEC Benchmark Workshop**. Austin, TX, USA: Springer Science & Business Media, 2009.

KONRADSSON, T. **ART and Dalvik performance compared.** UMEA Universitet 2015.

QEMU. **QEMU**. Disponível em <http://wiki.qemu.org/>. Acesso em: 15 out. 2016.

SARTOR, A L. **AndroProf GIT repositor**, 2014. <https://bitbucket.org/AndersonSartor/androprof-release>. Acesso em: 01 out. 2016.

SARTOR, A. L., CORREA, U. E BECK, A. C. **AndroProf: A Profiling Tool for the Android Platform**: Computing Systems Engineering (SBESC). 2013 III Brazilian Symposium on (pp. 23-28): IEEE, 2013.

SOURCE ANDROID. **ART and Dalvik.** Disponível em <https://source.android.com/devices/tech/dalvik/>. Acesso em: 03 out 2016.

SOURCE ANDROID. **The Android Source Code**, Disponível em <https://source.android.com/source/initializing.html>. Acesso em: 04 ago. 2016.

SPEC. **SPEC Benchmark Downloads.** Disponível em <http://www.spec.org/download.html>. Acesso em: 11 nov. 2016.

STATCOUNTER. **StatCounter**. Disponível em <http://gs.statcounter.com/#mobile+tablet-os- ww-monthly-201404-201604>. Acesso em: 06 nov. 2016.

VAN BOCKHAVEN, C. E VAN KERKWIJK, J. **Android Patching.** 2014. 42f. Projeto de pesquisa (Mestrado em Sistemas), University of Amsterdam, 2014.

WILTON, S., JOUPPI, N. **CACTI: An Enhanced Cache Access and Cycle Time Model**. IEEE Journal of Solid-State Circuits, vol. 31, no. 5, pp. 677-688, 1996.

YADAV, R. E BHADORIA, R. S. **Performance Analysis for Android Runtime Environment**. Communication Systems and Network Technologies (CSNT), 2015 Fifth International Conference on (pp. 1076-1079): IEEE. 2015.

APÊNDICES: TABELAS REFERENTES AO NÚMERO DE INSTRUÇÕES

APÊNDICE A – Tabela com o número de instruções, utilizada para gerar a Figura 6.1

No. Instr./VM	*FFT*	*SOR*	*Serial*	*RSA*	*SQLite*
Dalvik	10,936	64,251	0,069	6,733	10,144
Java	2,172	14,079	0,072	4,396	9,794

Unidade de medida: 10^9

APÊNDICE B – Tabela com o número de instruções, utilizada para gerar a Figura 6.4

No. Instr./Aplicação	*FFT*	*SOR*
Java	2,172	14,079
JNI	0,811	2,480

Unidade de medida: 10^9