Análise de execução de aplicações paralelas em grades móveis com restrições de processamento e bateria

Frederico Cassis Ribeiro Santos

Frederico Cassis Ribeiro Santos

Análise de execução de aplicações paralelas em grades móveis com restrições de processamento e bateria

Dissertação apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Mestre em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientador: Prof. Dr. Paulo Sergio Lopes de Souza

USP – São Carlos
Maio de 2016

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados fornecidos pelo(a) autor(a)

Santos, Frederico Cassis Ribeiro
S634a Análise de execução de aplicações paralelas em grades móveis com restrições de processamento e bateria / Frederico Cassis Ribeiro Santos; orientador Paulo Sergio Lopes de Souza. - São Carlos - SP, 2016.
64 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2016.

1. Dispositivos Móveis. 2. Estimativa do Consumo de Energia. 3. Grade Computacional Móvel. 4. Computação Paralela. I. Souza, Paulo Sergio Lopes de, orient. II. Título.

Frederico Cassis Ribeiro Santos

Analisys of the execution of parallel applications using a mobile grid environment

Master dissertation submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP, in partial fulfillment of the requirements for the degree of the Master Program in Computer Science and Computational Mathematics. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Paulo Sergio Lopes de Souza

USP – São Carlos
May 2016

Dedico este trabalho à minha família.

AGRADECIMENTOS

Agradeço primeiramente a minha esposa e meu filho que sempre me apoiam em todas as escolhas que faço e por me ajudarem a manter o foco mesmo convivendo à distância.

Agradeço ainda aos meus pais e irmão que sempre me apoiaram durante todo o mestrado.

Agradeço ao Prof. Dr. Paulo Sérgio Lopes de Souza por todo o conhecimento compartilhado em suas aulas, seminários, discussões do grupo e pela grande ajuda durante este projeto.

À Profa. Dra. Sarita Mazzini Bruschi por todo o auxílio e atenção que recebi durante o desenvolvimento deste projeto de mestrado assim como suas contribuições para o mesmo.

A todos os amigos que conheci tanto no LaSDPC quanto na USP incluindo os que vieram a morar comigo compartilhando bons momentos.

Também ao Prof. Dr. Marcos José Santana e Profa. Dra. Regina Helena C. Santana, as suas ótimas aulas.

Sou grato também a CAPES e ao CNPq pelo apoio financeiro durante todo o curso.

“A mente que se abre a uma nova idéia
jamais voltará ao seu tamanho original.”
(Albert Einstein)

RESUMO

SANTOS, F. C. R.. **Análise de execução de aplicações paralelas em grades móveis com restrições de processamento e bateria**. 2016. 64 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Existem atualmente diversas propostas para integração de dispositivos móveis em uma grade computacional, porém vários problemas são observados em tais ambientes. Esta dissertação mantém o foco em um problema, a restrição sobre a quantidade de energia despendida na execução das aplicações, ao utilizar esses dispositivos móveis como provedores de recursos em uma grade computacional que fornece processamento para aplicações paralelas. Para tanto, este trabalho propõe um método para estimar o consumo de energia das aplicações considerando que elas utilizam um determinado conjunto de operações as quais estão presentes na grande maioria das aplicações paralelas (operações matemáticas e alocação de memória). Com base no método proposto, dois dispositivos móveis foram estudados e foi criada uma representação do consumo de energia utilizando-se de métodos de regressão. Para validar os modelos, duas aplicações foram analisadas e o consumo de energia real foi comparado ao consumo estimado. O modelo criado apresentou resultados próximos ao medido, mostrando um aumento entre 6% e 14,24% em relação ao resultado medido.

Palavras-chave: Dispositivos Móveis, Estimativa do Consumo de Energia, Grade Computacional Móvel, Computação Paralela.

ABSTRACT

SANTOS, F. C. R.. **Análise de execução de aplicações paralelas em grades móveis com restrições de processamento e bateria**. 2016. 64 f. Dissertação (Mestrado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação (ICMC/USP), São Carlos – SP.

Nowadays, there are different proposals to integrate mobile devices in a computational grid, although several problems are introduces. This dissertation focus on the energy limitation problem when using mobile devices to provide resources, such as processing power to run parallel applications. It also proposes a method to estimate energy consumption for a task that needs to be executed in this environment. To achieve this goal two mobile devices were used as a test case and a representation of its energy consumption was created running benchmarks and using regression techniques. To validate the model created, two applications were executed and had the measured values compared to the estimated ones. The estimation showed a raise between 6 and 14.24 percent.

Key-words: Mobile Devices, Energy Consumption Estimation, Mobile Computational Grid, Parallel Computing.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AMPQ ... *Advanced Message Queuing Protocol*
DER ..... *Diagrama Entidade Relacionamento*
DSP ..... *Digital Signal Processor*
DVFS .... *Dynamic Voltage/Frequence Scaling*
GCC ..... *GNU Compiler Collection*
GDB ..... *GNU Debugger*
JIT ..... *Just in Time*
JSON .... *JavaScript Object Notation*
LCD ..... *Liquid Crystal Display*
OGSA .... *Open Grid Services Architecture*
PSS ..... *Proportional Set Size*
RISC ..... *Reduced Instruction Set Computing*
SDK ..... *Software Development Kit*
SOAP .... *Simple Object Access Protocol*
SOC ..... *System on Chip*
TCP ..... *Transmission Control Protocol*
UDP ..... *User Datagram Protocol*
WAN ..... *Wide Area Network*
WSDL ... *Web Services Description Language*

SUMÁRIO

CAPÍTULO

1

INTRODUÇÃO

1.1 Contextualização e Motivação

O surgimento do conceito de computação em grade aconteceu devido à existência de problemas científicos altamente paralelizáveis que necessitavam de muito esforço computacional para serem resolvidos. Inicialmente, as empresas combinavam diversos computadores em um cluster onde eram somados os recursos de todos os computadores para resolver esses problemas. Em outros casos, quando não se podia configurar um cluster para esse fim, era possível utilizar computadores que estavam em diferentes locais para formar uma grade computacional. Uma grade computacional combinava muitas vezes computadores de diversas empresas e compartilhavam seus recursos através de uma rede tipo *Wide Area Network* (WAN) ou em português, Rede de Longa Distância. O objetivo dessas grades computacionais era também de oferecer recursos para serem consumidos como se todos os computadores fossem apenas um. Diferentes abordagens de grade foram propostas inclusive com a integração de notebooks (PHAN; HUANG; DULAN, 2002). Hoje em dia os dispositivos móveis são muito comuns e sua popularidade vem crescendo de modo que a venda de dispositivos móveis ultrapassou a venda de computadores pessoais e continua crescendo ano a ano (GARTNER, 2015).

Embora esses dispositivos utilizem processadores cada vez mais potentes, atualmente os *tablets* e *smartphones* ficam na maior parte do tempo ociosos enquanto não estão sendo utilizados ou estão carregando. Devido a esse fato, existem diversas propostas para utilizar esses dispositivos em conjunto para resolver problemas computacionais ou então compor esses dispositivos em uma grade computacional móvel para prover recursos (BUSCHING; SCHILDT; WOLF, 2012). Uma grade formada por *smartphones* pode ser útil em diversos casos. Por exemplo na área de educação, alunos de uma universidade poderiam implementar um sistema que utilizasse seus celulares para testar algoritmos de criptografia ou processamento de sinais (KEHTARNAVAZ; PARRIS; SEHGAL, 2015) e estudar as características de cada um. Ainda, em países carentes de infraestrutura computacional, uma grade móvel poderia ser utilizada para proporcionar um

ambiente de cooperação para resolução de problemas. Finalmente, dispositivos móveis poderiam ser usados também para compor uma grade computacional convencional compartilhando recursos como espaço em disco, tempo de processamento e dados de sensores (SHEN, 2015).

A *Internet das Coisas* é um ambiente onde, gradativamente, todos os objetos estão conectados trocando informações. Muitos serviços utilizados nesse ambiente apresentam requisitos de baixa latência e necessitam consumir recursos como processamento e armazenamento. A *Fog Computing* é um novo paradigma que tem o objetivo de melhorar a eficiência desses sistemas deixando de enviar todos os dados coletados pelos dispositivos para a nuvem, diminuindo o tempo de reposta e a sobrecarga dos computadores em *datacenters*. Isso é possível ao utilizar os recursos disponíveis nos próprios dispositivos móveis (SARKAR; CHATTERJEE; MISRA, 2015) (DATTA; BONNET; HAERRI, 2015) (CIRANI *et al.*, 2015) (LIN; SHEN, 2015).

A introdução de dispositivos móveis em grades computacionais pode acontecer de duas maneiras diferentes. A primeira propõe o uso desses dispositivos apenas como consumidores de recursos ou interfaces para serviços. A segunda abordagem se justifica na capacidade de processamento ociosa desses dispositivos e propõe sua utilização como provedores de recursos para processamento paralelo, armazenamento distribuído ou fornecimento de informações de seus sensores e câmeras (RODRIGUEZ; MATEOS; ZUNINO, 2012).

Atualmente, a formação de uma grade computacional móvel é um assunto amplamente pesquisado, pois ao mesmo tempo que apresenta vantagens, também introduz novos problemas como por exemplo, a conexão parcial e instável de uma rede sem fio e a constante entrada e saída de nós no sistema. Um problema central nesse ambiente, é a limitação de processamento dos aparelhos e a energia consumida por esses dispositivos para contribuir com a grade.

Quando uma grade móvel é utilizada com a finalidade de prover recursos, as aplicações que serão executadas nessa grade precisam ser escalonadas de maneira a atender certos requisitos. Vários requisitos podem ser enumerados, tais como: ter o menor consumo de energia, ter suporte a tolerância a falhas, apresentar o menor tempo de resposta, entre outros. Particularmente, o estado da bateria e o consumo de energia necessário para a execução de uma aplicação são muito importantes pois podem determinar a disponibilidade ou não de um nó nesse ambiente.

No trabalho proposto por Luiz C. Borro (BORRO; ROCHA FILHO; BRUSCHI, 2014), o qual motivou o trabalho desenvolvido nesse projeto de mestrado, duas heurísticas foram propostas de modo a realizar o escalonamento das tarefas de uma aplicação minimizando o consumo de energia na grade como um todo, desde que o *deadline* da aplicação não fosse violado.

Dessa forma, para construir um ambiente de computação em grade móvel é muito importante considerar a utilização de técnicas que minimizem o consumo de energia em cada dispositivo móvel. Isso é possível de ser obtido quando se conhece o comportamento do consumo, o qual por sua vez depende do tipo de aplicação e também das características do dispositivo.

Por exemplo, um aparelho pode apresentar um consumo de energia maior do que outro para aplicações limitadas por processamento *(CPU Bound)* enquanto outro pode ser mais econômico em aplicações limitadas por entrada e saída *(I/O Bound)*. O trabalho de (BORRO; ROCHA FILHO; BRUSCHI, 2014) considera que as aplicações irão consumir uma determinada quantia de energia do dispositivo móvel, mas não detalha como esse valor foi obtido.

1.2 Objetivos

Como descrito na seção anterior, um dos principais problemas ao executar uma aplicação em uma plataforma móvel é o consumo de energia e a limitação da capacidade da bateria. É preciso que o programador da aplicação esteja ciente de qual tipo de código consome mais energia ou qual tipo de algoritmo é mais apropriado para um fim específico considerando outras métricas que não somente o tempo de resposta da aplicação.

O objetivo deste projeto é analisar a execução de aplicações em dispositivos móveis e definir uma metodologia para estimar o consumo de energia de aplicações que podem ser executadas em um ambiente de grade computacional móvel. Como a variedade de aplicações que podem ser executadas na grade móvel é muito grande, o presente trabalho restringe essas aplicações às que podem ser paralelizadas em uma grade computacional móvel. Esse tipo de aplicação geralmente envolve a resolução de problemas de natureza embaraçosamente paralela onde os dados utilizados em cada iteração são independentes e podem ser processados individualmente. É possível citar exemplos como algoritmos de força bruta, algoritmos genéticos, simulações computacionais, processamento de imagens e muitos outros.

1.3 Metodologia

Para que os objetivos fossem atingidos, primeiramente foi feita uma revisão da literatura considerando os trabalhos envolvendo o consumo de energia em dispositivos móveis. Em paralelo com a revisão da literatura, foi proposto um *middleware* para execução de aplicações paralelas em um ambiente de grade móvel, com o objetivo de verificar a viabilidade da proposta de se utilizar a grade móvel para execução de aplicações paralelas. Utilizando o *middleware* proposto, foi elaborado um estudo envolvendo um algoritmo paralelo de multiplicação de matrizes, o qual foi implementado utilizando duas estruturas de dados diferentes para representar as matrizes.

Após o desenvolvimento do *middleware*, foi proposta uma metodologia para calcular o consumo de energia de uma aplicação móvel. A metodologia proposta neste trabalho seguiu os seguintes passos:

Definição de *benchmarks* Identificação de programas que serviriam como *benchmarks* para serem executados em dispositivos móveis, os quais iriam fornecer informações para

moldar o comportamento desses aparelhos quando submetidos a uma determinada carga de trabalho.

Execução dos *benchmarks* Execução dos *benchmarks* e cálculo do consumo de energia médio considerando o desvio padrão em cada experimento.

Análise dos resultados Obtenção de equações que representam o consumo de energia de cada dispositivo utilizando técnicas de regressão.

Utilização dos dados Armazenamento dessas informações em um banco de dados para que um escalonador possa utilizar os dados para estimar o consumo de energia de um dispositivo

De modo a validar a metodologia proposta, o consumo de energia de duas aplicações foi medido através da execução das mesmas nos dispositivos móveis e foi também estimado utilizando as informações armazenadas no banco de dados.

O objetivo da metodologia é ser genérica, no sentido de que outros *benchmarks* podem ser incluídos e também estes podem ser executados em outros dispositivos móveis, permitindo representar através de equações o consumo de energia de outras operações e/ou métodos.

1.4 Organização

Esta dissertação está organizada da seguinte maneira: Nos Capítulos 2 e 3 se encontram a revisão bibliográfica onde são apresentados conceitos sobre a teoria de sistemas distribuídos, middlewares para computação em grade e consumo de energia para aplicações desenvolvidas nas plataformas Java e Android. Em seguida, o Capítulo 4, apresenta como o projeto foi desenvolvido mostrando as ferramentas utilizadas para a realização dos experimentos e também os programas desenvolvidos como casos de teste. Esse Capítulo mostra também o processo utilizado para validar os dados obtidos nos experimentos. O Capítulo 5 apresenta os dados obtidos nos experimentos e os gráficos obtidos através da regressão dos dados. Finalmente, o Capítulo 6 apresenta as conclusões sobre o projeto, contribuições do mesmo e os projetos futuros que podem contribuir com o assunto pesquisado neste trabalho.

CAPÍTULO

2

GRADES COMPUTACIONAIS MÓVEIS

2.1 Considerações Iniciais

Atualmente o conceito de grades computacionais móveis, utilizado nesse projeto de mestrado, está bastante difundido, principalmente devido a utilização de *smartphones* pela maior parte da população. Os *smartphones* são exemplos de dispositivos móveis que podem pertencer a uma grade móvel fornecendo desde informações de sensoriamento até processamento. Como o conceito de grade móvel tem como base o conceito de grade computacional, inicialmente neste capítulo são apresentados os conceitos de grades computacionais, tipos de grades existentes e o padrão de arquitetura de uma grade de serviços. Posteriormente, os conceito de grade móvel e de *middleware* para computação em grade móvel também são apresentados e por fim, exemplos de utilização de grade móvel são apresentados através de projetos de computação voluntária.

2.2 Grades Computacionais

No passado, quando a complexidade das aplicações científicas aumentou e não havia recursos suficientes para a compra de uma máquina massivamente paralela, os cientistas começaram a procurar um modo de conectar diferentes máquinas, através de uma rede local, para criar um computador mais potente, podendo assim compartilhar os recursos dessas máquinas. Essa combinação é chamada de *cluster* e se caracteriza por ser um sistema homogêneo conectado por redes de alta velocidade (PFISTER, 1998).

Infelizmente nem todos podiam contar com recursos para comprar o seu próprio *cluster* de computadores e, com isso, as empresas tinham que combinar suas máquinas em diferentes locais, construindo assim um ambiente de computação em grade e criando o conceito de organizações virtuais. Pessoas que pertencem a mesma organização virtual têm direitos de acessar recursos e serviços das máquinas pertencentes a essa organização. É possível que muitos com-

putadores presentes em uma organização virtual estejam em *datacenters* ou *clusters* privados de uma empresa. Além do poder de processamento, outro serviços que englobam armazenamento, equipamentos especiais e sensores (TANENBAUM, 2007) podem ser oferecidos. Para a implementação de um sistema desse tipo, uma arquitetura de software foi proposta por Foster (2002).

Desse modo, uma grade computacional é formada pela conexão de computadores ou dispositivos que são utilizados para fornecer serviços e recursos a uma organização virtual. Em contrapartida a um *cluster*, a grade se diferencia por ser fracamente acoplada ou seja, em uma grade, os computadores podem estar em locais geograficamente diferentes e podem constituir um ambiente heterogêneo onde cada máquina pode ter uma arquitetura diferente ou executar tipos diferentes de sistema operacional (BERRY DJAOUI, 2006).

Uma das principais estratégias para construir um ambiente de grade computacional é o de utilizar uma camada de software, chamada de *middleware*, para abstrair e simplificar alguns problemas. Essa camada adiciona transparência no sistema, que é uma característica importante dos sistemas distribuídos.

Uma grade computacional deve ocultar o fato de que processos e recursos estão fisicamente distribuídos em computadores diferentes e podem ser categorizadas em diferentes tipos (KRAUTER; BUYYA; MAHESWARAN, 2002):

Grade de Processamento: Uma grade de processamento tem como objetivo oferecer uma infraestrutura de processamento distribuído, ou seja, fornecer poder computacional de modo que qualquer usuário do sistema possa utilizar o processamento de todas as máquinas da grade. Esse tipo de grade é geralmente utilizado para resolução de problemas científicos e que tem uma solução da qual pode ser paralelizada.

Grade de Dados: Este tipo de grade fornece uma infraestrutura distribuída para armazenamento de dados. Muitos sistemas de banco de dados distribuídos utilizam essa abordagem para aumentar a disponibilidade e espaço de armazenamento.

Grade de Serviços: Uma grade de serviços fornece um ou mais serviços específicos que geralmente seriam impossíveis de ser alcançados utilizando-se de apenas um computador. Esses serviços podem variar de acordo com os requisitos e podem envolver a integração entre os usuários e as aplicações.

2.3 Open Grid Services Architecture e Globus Toolkit

Em meados de 2000, para facilitar o desenvolvimento de aplicações distribuídas em um ambiente de grade computacional, Ian Foster et. al (FOSTER, 2002) propuseram a padronização desses serviços em um ambiente de grade computacional formado por organizações virtuais.

A *Open Grid Services Architecture* (OGSA), é uma arquitetura que define mecanismos para criação, nomeação e descoberta de serviços nesse ambiente. A arquitetura proposta se utiliza da linguagem *Web Services Description Language* (WSDL), que é uma linguagem utilizada para descrição de interfaces de serviços nesse ambiente (FOSTER, 2002) (FOSTER; KESSELMAN; TUECKE, 2001). Embora essa proposta apresente uma definição de como construir uma grade de computação distribuída e de como os computadores podem se comunicar, a especificação não define detalhes de implementação, tais como respostas para um serviço específico ou questões sobre qual tecnologia deve ser utilizada.

Baseada nas ideias propostas anteriormente foi publicada em 2005 a primeira versão da especificação do padrão OGSA (KISHIMOTO FUJITSU, 2005). Esse documento foi desenvolvido por membros do *Open Grid Forum*, que é uma comunidade de usuários, desenvolvedores e empresas interessadas na padronização da computação em grade. O padrão é baseado na troca de mensagens utilizando o protocolo *Simple Object Access Protocol* (SOAP), que é um protocolo de camada de aplicação, e independe do protocolo de transporte utilizado para isso. A interoperabilidade e portabilidade nessa proposta é alcançada utilizando *webservices*. Dessa forma, os nós podem se comunicar e compartilhar recursos computacionais. Posteriormente, a versão 1.5 descreveu responsabilidades da especificação como serviços de infraestrutura, serviços de segurança e serviços de informação.

Uma implementação de um sistema computacional em grade é o *Globus Toolkit* (FOSTER, 2005). Esta implementação fornece um conjunto de programas, serviços e bibliotecas para a construção de um ambiente seguindo o padrão OGSA. É um projeto de código aberto desenvolvido desde 1996 e atualmente está na versão 5. O software fornece implementações de serviços focados na administração de uma infraestrutura, ferramentas para construção de novos *webservices* em linguagens de programação como Java, C e Python, uma infraestrutura robusta e segura e bibliotecas e programas de linha de comando para acesso a todos os serviços do sistema.

O *Globus Toolkit* ainda proporciona meios para que os clientes possam ser autenticados e utilizem o sistema de forma segura. Provê também meios para que exista um controle de permissões e descoberta de novos serviços.

2.4 Grades Computacionais Móveis

É possível que dispositivos móveis, tais como sensores, façam parte de uma grade computacional fornecendo informações de monitoramento ou outros tipos de serviços (CHOI; JEON; PARK, 2010). Uma grade computacional móvel se caracteriza por agregar dispositivos móveis conectados entre si por uma rede sem fio com o objetivo de fornecer algum serviço específico, como por exemplo monitorar a temperatura de um ambiente ou umidade do ar. Essa rede de sensores poderia ser integrada junto a uma grade computacional comum fornecendo essas informações por meio de um *broker* (ISAIADIS; GETOV, 2005). Não apenas sensores,

mas também dispositivos móveis, tais como *smartphones* e *tablets*, poderiam fazer parte de uma grade computacional fornecendo seu poder de processamento.

A facilidade na implementação de aplicativos para dispositivos móveis pode auxiliar a criação de uma nova geração de computação em grade. No entanto, novos problemas são criados ao introduzir esse tipo de dispositivo para executar aplicações distribuídas. Dentre esses problemas podemos citar o de como organizar os nós para que troquem mensagem na rede, como realizar a descoberta de recursos em uma topologia onde cada nó pode ficar indisponível a qualquer momento, como realizar o escalonamento de tarefas ciente do consumo de energia e o problema de dividir tarefas para serem processadas em um ambiente heterogêneo onde cada dispositivo pode utilizar diferentes plataformas de sistema operacional (RODRIGUEZ; ZUNINO; CAMPO, 2011).

Dentre os problemas citados anteriormente, pode-se destacar o problema do consumo de energia. A quantidade de energia em um dispositivo móvel em um dado momento e a quantidade de energia que uma tarefa necessita para ser executada são cruciais para que o sistema funcione de forma satisfatória. O trabalho do escalonador em uma grade computacional é justamente fazer a atribuição das tarefas nos processadores de acordo com essas restrições. Ainda, a energia total necessária para resolver um problema computacional na grade móvel não pode ser maior do que a energia somada disponível em todos os nós. O estudo relacionado à viabilidade da execução de aplicações nesse cenário está no escopo deste projeto.

2.5 *Middlewares* para Computação em Grade Móvel

Um *middleware* pode ser definido como uma camada de software no qual o propósito é mascarar a heterogeneidade e proporcionar um modelo de programação conveniente para que programadores possam facilmente desenvolver aplicações nesse ambiente (COULOURIS, 2012). O software é representado por processos ou objetos em um conjunto de computadores que interagem entre si para um objetivo específico.

Para criar um alto nível de abstração, um *middleware* deve cumprir com alguns requisitos funcionais e não funcionais. Alguns desses requisitos funcionais são (BRUNEO; PULIAFITO; SCARPA, 2007):

1. *Comunicação:* O *middleware* tem que garantir a troca de dados de forma confiável entre os nós do sistema distribuído;

2. *Coordenação:* O *middleware* deve coordenar as atividades executadas por todos os nós;

3. *Descoberta de Serviços:* Deve ser possível descobrir recursos e serviços disponíveis no sistema;

4. *Posição e profile do Usuário:* O *middleware* deve filtrar os serviços e recursos que devem ser escolhidos de acordo com as preferências, hábitos e localização do usuário. Essa última consideração é importante em sistemas no qual a latência ou o atraso pode comprometer a qualidade do serviço;

5. *Requisitos de Qualidade de Serviço:* O *middleware* deve se preocupar em fornecer um serviço livre de erros e que cumpra com o tempo de resposta esperado;

6. *Balanceamento de Carga:* O *middleware* deve ser capaz de lidar com frequentes desconexões comuns em um ambiente wireless onde os nós apresentam uma conexão parcial no sistema.

Alguns requisitos não funcionais impõem restrições que são críticas para determinar o sucesso ou não de um *middleware* para um ambiente de grade móvel. De acordo com (BRUNEO; PULIAFITO; SCARPA, 2007), (FRANKE; KOWALEWSKI; WEISE, 2012), e (POPESCU *et al.*, 2011), alguns desses requisitos são:

Interoperabilidade: Devido a heterogeneidade dos Sistemas Operacionais em cada nó de um sistema distribuído, a representação dos dados em cada um desses sistemas pode ser diferente. Isso deve ser antecipado pelo *middleware* para proporcionar um ambiente confiável de comunicação;

Confiança: Requisito que especifica a capacidade de um *middleware* móvel para operar sem falhas e manter um nível específico de desempenho quando utilizado em condições normais durante um período de tempo (MAIRIZA; ZOWGHI; NURMULIANI, 2010);

- **Disponibilidade:** O *middleware* precisa manter os serviços disponíveis mesmo quando os nós estão se movendo entre células de conexão;
- **Recuperabilidade:** O *middleware* precisa recuperar a sessão do usuário quando ocorrer algum problema de desconexão;

Desempenho: Requisito que especifica a capacidade do software em proporcionar um desempenho apropriado relativo a quantidade de recursos necessários para atender funcionalmente dentro de condições esperadas (MAIRIZA; ZOWGHI; NURMULIANI, 2010);

- **Utilização de Recursos:** O *middleware* precisa otimizar o uso de recursos nos dispositivos (como i.e., memória, energia, armazenamento e CPU);
- **Perda de Dados:** O *middleware* precisa evitar a perda de dados quando o sinal for perdido ou ocorrer interferências na conexão;

Escalabilidade: Requisito que descreve o quanto uma solução é boa para um problema que mesmo que cresca, ainda se torne viável (GORTON, 2006);

Balanceamento de Carga: O *middleware* deve direcionar novas requisições de tal maneira que não comprometa o desempenho do sistema;

Transparência de Localização: O usuário deve ser capaz de utilizar o software sem que o mesmo perceba qual o local real da utilização dos recursos e execução de tarefas;

Segurança: Requisitos que se preocupam em prevenir que usuários não autorizados acessem e comprometam o sistema, violem ou roubem dados (MAIRIZA; ZOWGHI; NURMULIANI, 2010);

Controle de acesso: O *middlware* precisa verificar a identidade dos usuários, suas permissões no sistema e fazer com que cada usuário acesse apenas os recursos aos quais tem permissão;

Autenticação: O *middleware* precisa verificar a identidade real dos usuários e componentes de software para proteger o sistema de nós mal intencionados;

Confidencialidade: O *middleware* deve proteger os dados de serem lidos por usuários não autorizados;

Integridade: O *middleware* precisa assegurar que os dados trocados entre os nós não sejam violados, ou alterados;

Flexibilidade: É quando um sistema pode ser modificado para ser utilizado em aplicações ou ambientes diferentes daquele para o qual foi desenvolvido inicialmente (IEEE. . . , 1990);

Extensibilidade: O *middleware* deve ser extensível para adicionar suporte a novos componentes de hardware e novas configurações;

Portabilidade: O *middleware* deve ser capaz de operar em diversas plataformas diferentes;

Adaptabilidade: O sistema pode ter que reconhecer funcionalidades em um dispositivo e adaptar seus serviços de acordo. Esse requisito também pode ser descrito como *context-awareness*.

2.6 MobiGrid

Um dos primeiros *middleware* desenvolvido para atuar em um ambiente de grade móvel formado por *smartphones* foi o MobiGrid (MASINDE; BAGULA; NDEGWA, 2010). Esse *middleware* foi proposto para ser utilizado em países em desenvolvimento que sofrem com a falta de infraestrutura. Nesse caso, uma grade formada por *smartphones* seria bastante útil para proporcionar computação distribuída.

MobiGrid foi desenvolvido baseado em um modelo cliente/coordenador no qual um dos dispositivos da rede sempre estaria atuando como o coordenador do sistema distribuído. Ele foi

projetado para cumprir com alguns requisitos. Primeiramente, como um sistema distribuído o *middleware* proporciona um meio para que todos os nós estabeleçam conexões entre si. Ainda, um número finito de comandos entendido por todos os nós é definido para que a troca de mensagens ocorra de forma correta. Dentre essas mensagens de controle, é possível que um nó descubra quais outros nós estão na grade e quais serviços cada nó oferece. Caso haja algum problema com o nó coordenador, MobiGrid conta com um algoritmo de eleição para determinar que um novo nó atue como coordenador do sistema. Esse algoritmo leva em conta restrições sobre a quantidade de bateria em um dispositivo.

Outra característica importante desse sistema é que o mesmo implementa um serviço de descoberta que é capaz de monitorar todos os nós ativos ou inativos em um dado momento. O *middleware* implementa todas essas funcionalidades e tem a habilidade de se recuperar caso aconteçam falhas na comunicação, muito comuns nesse ambiente volátil de conexão sem fio. MobiGrid foi projetado para suportar qualquer dispositivo que utilize o Sistema Operacional SymbyanOS, que era o mais popular na época. Foi implementado na linguagem de programação Python e testado em dois modelos de *smartphones* diferentes, sendo eles, Nokia E63 e Nokia N95.

O *middleware* conta com dois módulos: *Local Server* e *Coordinator Server*. Para construir uma grade de *smartphones* é preciso que as aplicações sejam instaladas em todos os nós e que a *Coordinator Server* seja instalado nos nós que poderão agir como coordenador do sistema.

MobriGrid utiliza o paradigma de passagem de mensagem que são enviadas e recebidas ao coordenador representadas como objetos *JavaScript Object Notation* (JSON), utilizando o protocolo da camada de transporte *User Datagram Protocol* (UDP). Em sua última versão o *middleware* pode reconhecer apenas coordenadores em um ambiente que esteja utilizando a máscara de rede 255.255.255.0 e nenhum esforço foi ainda realizado para proporcionar um ambiente seguro.

2.7 Grid Anywhere

Um projeto desenvolvido em uma tese de doutorado no Laboratório de Sistemas Distribuídos e Programação Concorrente (LaSDPC), ICMC/USP, chamado Grid Anywhere (TEIXEIRA, 2012) propõe um ambiente Java onde qualquer dispositivo que suporte essa plataforma poderia fazer parte da grade computacional. O objetivo desse projeto é proporcionar um ambiente para computação em grade e compartilhamento de recursos entre computadores e receptores de televisão digital. O Grid Anywhere utiliza o protocolo *SOAP* para estabelecer a comunicação entre os nós e sua especificação de arquitetura, permite que novas formas de transporte de mensagens *SOAP* sejam adicionadas ao sistema.

Diferente do MobiGrid, proposto para a plataforma SymbyanOS, o Grid Anywhere utiliza serialização de objetos para proporcionar a migração de código. Assim, o código a ser

executado não precisa estar compilado previamente no nó utilizado como recurso da grade. A Figura 1 mostra a serialização de um objeto Java. Primeiramente, o objeto é transformado em um *array* de bytes que contém todos os outros objetos e seus dados primitivos. Depois disso, o objeto pode ser armazenado em um banco de dados, na memória ou enviado pela rede.

O *middleware* se preocupa ainda com a segurança do sistema fornecendo um ambiente seguro chamado de Sam Dog. Para compartilhar recursos em uma grade computacional, esse ambiente fornece mecanismos de gerenciamento de políticas de segurança. As regras sobre as permissões de usuários são armazenadas em diferentes camadas do sistema e podem ser herdadas de domínios e subdomínios do qual um usuário pode pertencer. O sistema suporta a criação de grupos que podem pertencer a apenas um subdomínio. Quando um usuário é atribuído a um grupo, o mesmo é também atribuído ao subdomínio deste grupo. Para integrar o ambiente Sam Dog e assegurar que as regras de permissão sejam cumpridas, o *middleware* sobrescreve o *AcessController* por uma implementação própria, o *SamController*. Na plataforma Java, o *AccessController* tem o objetivo de fornecer regras sobre a execução de um código pela Máquina Virtual Java.

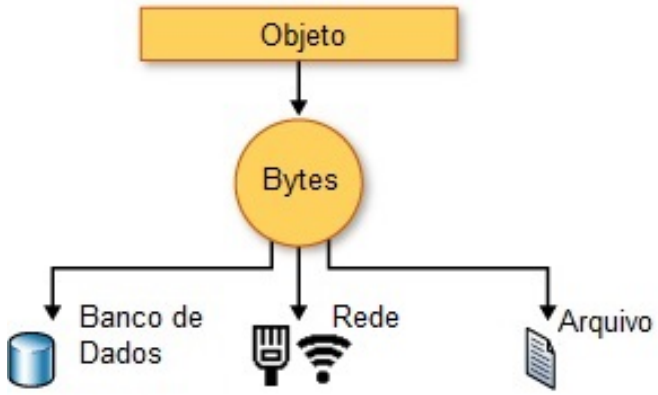

Figura 1 – Serialização de um Objeto [Adaptado de msdn.microsoft.com]

O *middleware* proposto foi totalmente implementado em dois módulos: Sam Manager e Sam Controller. O primeiro permite ao usuário o gerenciamento das políticas do sistema, enquanto o segundo consiste no ambiente seguro de execução de aplicações Java que utiliza as políticas de controle definidas pelo primeiro.

2.8 Outros Middlewares Móveis

Outros *middleware* com diferentes objetivos foram propostos para ambientes com requisitos de mobilidade. Uma dessas propostas apresentou um *middleware* para uma rede de sensores que se preocupa com restrições de consumo de energia e heterogeneidade de hardware (KAMAL; RAZZAQUE; HONG, 2011). Varaprasad e Raju (2005) apresentaram um *framework*, baseado em uma arquitetura flexível do tipo *cliente-servidor* que utiliza um *middleware* para proporcionar flexibilidade e escalabilidade para configuração e comunicação em um ambiente

heteregêneo, formado por dispositivos conectados por uma rede wireless. Nguessan *et al.* (2008) propuseram um modelo de segurança que implementa um mecanismo de autenticação e controle de acesso para agentes móveis utilizando uma implementação de espaço de tuplas para esse fim.

Verbelen *et al.* (2012) apresentaram AIOLOS, um *framework* para Android. Esse fornece um serviço para que o programador escolha alguns métodos a serem executados remotamente. O AIOLOS então decide, em tempo de execução, qual desses métodos serão executados remotamente, considerando o contexto atual do dispositivo e assim, migra o código para ser executado em um servidor. Para cada chamada de método, AIOLOS decide se a chamada será executada local ou remotamente usando um modelo baseado em histórico da velocidade de rede e processamento desse servidor. Para que o *middleware* trabalhe de forma transparente, o mesmo fornece todas as ferramentas necessárias para que as aplicações se beneficiem desse *framework* anotando métodos Java de uma aplicação. De forma similar, Makki *et al.* (2011) e Ashmore e Makki (2011) apresentam IMISSAR, que adiciona em uma aplicação o suporte a *context-awareness*, que é a capacidade de se adaptar de acordo com o ambiente. Esse *middleware* utiliza uma base da dados inteligente para lidar com erros e suporta a reconfiguração automática de aplicações para que códigos locais sejam executados remotamente. O principal objetivo do IMISSAR é o de proporcionar a auto-reconfiguração para aplicações sem grandes alterações em um código existente.

Recentemente Badam, Fisher e Elmqvist (2015) propuseram um *framework* para construção de um ambiente heterogêneo de computação ubíqua nomeado Munin. Esse *framework* de comunicação proporcionou a interoperabilidade entre dispositivos móveis, sendo eles, *smartphones*, *tablets* e telas de *Liquid Crystal Display* (LCD) conectados em uma rede *peer-to-peer*. O *framework* foi dividido em três camadas: *Shared State Layer* responsável por proporcionar a comunicação entre os nós utilizando o padrão publicar/assinar; *Service Layer* sendo uma interface com diferentes serviços como entrada de dados, renderização e processamento de acordo com as características de cada dispositivo; *Visualization Layer* com componentes de alto nível incluindo um banco de dados distribuído utilizado para replicar dados entre os nós. Pagel e Carlson (2015) apresentaram um *middleware* para auto integração de dispositivos móveis conectados no mesmo ambiente. Utilizando-se de um sistema de descoberta os dispositivos foram capazes de trocar mensagens e compartilhar recursos com baixa utilização de processamento e memória.

Um outro *middleware* proposto para compartilhamento de recursos em um ambiente móvel foi proposto por Shih, Wang e Chang (2015). Neste trabalho, dispositivos como *smartphones* e computadores remotos disponíveis no ambiente foram caracterizados de acordo com o atraso na resposta de requisições, tempo de resposta esperado e capacidade de processamento. O *middleware* proporcionou a alocação de recursos entre esses dispositivos onde foram analisados diferentes algoritmos para o escalonamento de tarefas. Perera *et al.* (2014) propuseram ainda que os dados coletados por sensores, utilizados no contexto da Internet das Coisas, necessitam primeiro de ser tratados antes de enviados para processamento em uma nuvem. Para isso, foi

apresentado, neste trabalho, um *middleware* para dispositivos móveis responsável por realizar o processamento prévio desses dados sem a necessidade de desenvolver nenhuma outra aplicação. O sistema proposto foi desenvolvido para a plataforma Android.

2.9 Projetos colaborativos que Utilizam Processamento Paralelo em Dispositivos Móveis

Existem atualmente diversos projetos que utilizam computação colaborativa e processamento paralelo. O objetivo desses projetos é formar uma grade computacional para utilizar os recursos de computadores e outros dispositivos quando os mesmos não estiverem sendo utilizados. A seguir, são descritos alguns desses projetos.

2.9.1 Berkeley Open Infrastructure for Network Computing

Conhecido como *BOINC*, o *middleware* para computação voluntária em grade teve sua primeira versão liberada ao público em 2002. O sistema proporciona meios para que seja possível integrar novos projetos facilmente utilizando sua plataforma e conta com servidores e clientes que se comunicam para processar as tarefas (ANDERSON, 2004). Alguns projetos utilizam ainda a plataforma *CUDA* desenvolvida pela empresa *NVIDIA* para utilizar Unidades de Processamento Gráfico como um recurso da grade computacional. Em Janeiro de 2016 existem 58 projetos em atividade utilizando essa plataforma (BOINCSTATS, 2016).

2.9.2 SETI@Home

Um projeto que foi pioneiro na questão da computação voluntária ao fazer com que usuários doassem tempo de processamento de seu computador ou dispositivo móvel para executar uma tarefa em comum foi o SETI@Home. O problema computacional que o projeto tenta resolver é o de procurar sinal de vida inteligente fora do planeta terra. Como entrada o sistema recebe dados que foram colhidos por um telescópio e utilizando um algoritmo para detectar ondas eletromagnéticas emitidas em um padrão não existente no espaço, o que comprovaria que a onda foi emitida por vida inteligente. Este projeto foi um dos principais motivos da criação do *BOINC* e permanece utilizando a plataforma até hoje (ANDERSON *et al.*, 2002).

2.9.3 BoincSimap / Samsung Power Sleep

Outro projeto interessante que utiliza a infraestrutura *BOINC* é o BoincSimap. Este projeto utiliza o processamento distribuído para compor um banco de dados contendo informações sobre similaridades entre proteínas. Trata-se de um problema de natureza altamente paralelizável onde a entrada é a codificação das sequências de aminoácidos. O algoritmo, chamado de *FASTA* é utilizado para buscar o alinhamento das sequências que é uma forma de organizar essas proteínas

para identificar regiões similares e assim descobrir relações entre as proteínas contidas na base de dados *bsimap*. Essa base de dados chamada de *SIMAP* é de público acesso a usuários e pesquisadores principalmente da área biológica. Outro fato interessante é que além da versão disponível para dispositivos móveis do sistema *BOINC*, existe uma aplicação conhecida como Samsung Power Sleep que utiliza o processamento do *smartphone* ou *tablet* Samsung®para colaborar com o projeto e pode ser instalado em qualquer dispositivo da marca (LIU *et al.*, 2014).

2.9.4 Folding@home

Outro projeto disponível para a plataforma Android, porém não implementado utilizando a estrutura do *BOINC* é o Folding@Home. Criado pela universidade de Stanford o projeto visa entender o comportamento das proteínas presentes no corpo dos seres vivos. Diferente do projeto citado anteriormente, o projeto não foca em apenas um problema computacional. Ao invés disso, diversos algoritmos e implementações diferentes foram testadas e o projeto continua a evoluir de acordo com as necessidades e as novas ideias de pesquisa nessa área. Além do cliente Android o sistema conta também com um cliente web que funciona no navegador Chrome e um cliente para o video-game PlayStation 3 (BEBERG *et al.*, 2009).

2.9.5 Outros Projetos

Existem ainda diversos outros projetos que fazem uso da computação voluntária. Muitos deles utilizam a infraestrutura enquanto outros implementam seu próprio mecanismo para obter um modo eficiente de compartilhar recursos geograficamente distribuído. Enquanto alguns projetos têm seu foco na área da biologia, como os citados anteriormente, outros têm como objetivo resolver problemas matemáticos como é o caso do *Background Pi*. Esse projeto visa calcular o número *PI* com a maior precisão possível. Outro projeto nessa área é o *Twin Prime Search* que tenta encontrar números primos grandes. Ainda, existem diversos projetos que visam testar a eficiência de métodos de criptografia como *Moo! Wrapper*, *Distributed.net* e *DistrRTgen*. Em outras áreas ainda existem diversos projetos que vão desde a área citada até mineração de dados distribuída e aprendizado de máquina.

2.10 Considerações Finais

As grades computacionais surgiram diante da necessidade de compartilhar recursos que estavam geograficamente distantes. As grades fornecem principalmente processamento, local para armazenamento de dados ou outros serviços. Algumas facilidades foram introduzidas para que as grades pudessem ser criadas e utilizadas, resultando em padrões e *middleware*.

Com a disseminação de dispositivos móveis tais como *smartphones*, a ideia de agregar esses dispositivos em uma grade computacional móvel, seja para fornecer informações de sensores ou processamento, se tornou bastante interessante. Porém, essa ideia acarretou em

outros desafios e um dos principais é com relação ao consumo de energia das aplicações que serão executadas nos dispositivos móveis. Esse é o tópico a ser tratado no próximo capítulo.

CAPÍTULO

3

CONSUMO DE ENERGIA

3.1 Considerações Iniciais

O objetivo deste capítulo é apresentar conceitos sobre a arquitetura atual mais comum em *smartphones*, *tablets* e computadores portáteis e descrever pontos importantes no funcionamento de consumo de energia desses dispositivos como travas de energia e gerenciamento dos recursos pelo sistema operacional Android.

3.2 Evolução dos Dispositivos Móveis

Há cinco anos o mundo *mobile* que conhecemos hoje, onde as pessoas estão conectadas a internet o tempo todo trocando mensagens ou compartilhando mídias, era algo longe da realidade. Os celulares e outros dispositivos móveis tinham poucas funcionalidades comparados aos *smartphones* atuais e as aplicações deixavam muito a desejar em questões de funcionalidades. Essa evolução foi apenas possível devido a melhorias no hardware desses aparelhos e de adaptação das aplicações e do sistema operacional visando sempre minimizar o consumo de energia, obtendo assim um desempenho satisfatório.

As primeiras gerações de celulares utilizavam exclusivamente processadores *Digital Signal Processor* (DSP). Esse tipo de processador proporciona um menor tempo de resposta e consumo de energia para algoritmos de processamentos de sinais, os quais geralmente apresentam um grande número de operações matemáticas que devem ser executadas rapidamente e repetitivamente. Um *DSP* recebe um sinal continuo analógico e passa esse sinal por um filtro analógico para remover as frequências mais altas. Em seguida é realizado o *sampling* que é a amostragem do sinal a cada intervalo de tempo, ou seja, um valor para a amplitude da onda para determinado instante. Cada um desses valores é convertido em bits para que possam ser processados. Finalmente o sinal digital é transformado em sinal analógico novamente utilizando-se de um conversor digital-analógico como mostra a Figura 2. Esse tipo de processador ainda é

utilizado nos *smartphones* atuais junto com outros componentes para o processamento de sinais (SINGH; JAIN, 2014). Um exemplo é o processador *Snapdragon* da fabricante *Qualcomm*.

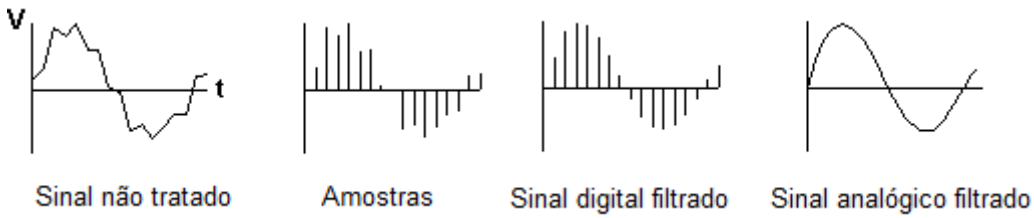

Figura 2 – Processamento de sinal por um processador DSP [Adaptado de cs.indiana.edu]

Para construir um dispositivo mais econômico é possível também utilizar um projeto com *System on Chip* (SOC), que consiste em utilizar diversos componentes integrados em uma mesma placa. É possível ainda, diminuir a voltagem de todo o chip, minimizando assim o consumo de energia do dispositivo móvel (AGUILERA; GALLEGO; SILVA, 2014).

3.3 Arquitetura Processador ARM

De todas as arquiteturas de processadores utilizadas na construção de dispositivos móveis e embarcados a mais utilizada atualmente é a ARM. A arquitetura ARM compõe uma família de processadores *Reduced Instruction Set Computing* (RISC) e possui um conjunto de instruções para todos os processadores *ARM* existentes hoje em dia. Devido às crescentes necessidades impostas principalmente por dispositivos embarcados, esse tipo de processador vem evoluindo para cumprir os requisitos de desempenho, consumo de energia e tamanho físico do componente. Como todos os processadores *RISC*, a arquitetura utiliza um registrador *load/store* onde os dados são processados. Utiliza também um tipo de endereçamento simples para a memória já que o acesso a memória é determinado por instruções e valores armazenados em registradores (ARM, 2015).

Apesar de inicialmente existirem apenas processadores ARMs de 32 bits, a arquitetura *ARMv8* apresenta diversas implementações para processadores ARM de 64 bits proporcionando um aumento de desempenho em alguns casos porém mantendo a principal característica de baixo consumo de energia. Alguns exemplos desses processadores são *Cortex-A72*, *Cortex-A57* e *Cortex-A53*. Outra característica melhorada nesses últimos hardware citados é o *pipelining* de instruções. Diferentes das versões anteriores que apresentavam um *pipelining* de três estágios houve uma melhoria na lógica para previsão proporcionando um nível maior de *pipelining* de instruções.

3.4 Consumo de Energia em Dispositivos Embarcados

Analisando o consumo de energia em dispositivos embarcados, é possível perceber que as instruções em linguagem de máquina geradas a partir do código fonte de um programa estão diretamente ligadas ao consumo de energia no hardware. A camada de software envolvida em um sistema computacional constituída pelo conjunto de instruções executadas pelo processador, a maneira que os dados são representados em um programa, a maneira como um algoritmo foi implementado e arquitetura do sistema podem influenciar em até 80% do consumo de energia total (LUO *et al.*, 2009).

Simunic *et al.* (2000) propuseram uma ferramenta de avaliação de desempenho utilizada para melhorar o algoritmo de decodificação do formato de áudio MPEG Layer III. Nesse trabalho, foram realizadas otimizações nos algoritmos de decodificação e na representação dos dados utilizados. Fei *et al.* (2004) mostraram também que otimizações no núcleo do Sistema Operacional pode diminuir o consumo de energia do sistema em 23,8%, em média. Esse trabalho explorou a interface do Sistema Operacional e a característica dos processos analisando seu código fonte.

Zhou *et al.* (2009) mostraram também que as instruções presentes em um código de um programa podem ser utilizadas para otimizar o consumo de energia do mesmo. Isso foi possível utilizando um debugger de código C do Linux chamado de *GNU Debugger* (GDB) adaptado para calcular o consumo de energia entre breakpoints inseridos no código. Essa mesma abordagem, de caracterizar cada instrução ao seu consumo de energia, pode ser utilizada para calcular o consumo total de energia de um programa (BAZZAZ; SALEHI; EJLALI, 2013) quando executado em um processador específico. Ainda, ao analisar um algoritmo é possível, com base em suas instruções em assembly, melhorar sua implementação para minimizar o consumo de energia.

Outras otimizações que minimizam o tempo de execução e gasto de energia são automaticamente realizadas por compiladores, essas otimizações são realizadas ao compilar o código fonte. Muitas dessas alterações no código é realizada para casos bem conhecidos na literatura e tem objetivo de eliminar operações não necessárias como, por exemplo, *loop unrolling* que elimina as variáveis de controle que são incrementadas a cada iteração de um *loop* copiando o código existente dentro do *loop* sucessivamente. Algumas otimizações mais utilizadas do compilador *GNU Compiler Collection* (GCC) são descritas em seu manual disponível online (FOUNDATION, 2013).

Em outros casos, como em algoritmos e representação de dados específicos de uma aplicação, esse tipo de otimização se torna impossível de ser realizada automaticamente. O compilador Java, por exemplo, não realiza quase nenhuma otimização. Elas são feitas pelo compilador *Just in Time* (JIT) que interpreta as instruções de *bytecode* Java e as transforma em instruções de máquina de acordo com a plataforma em tempo de execução. Esse compilador *JIT* está presente dentro da implementação da Máquina Virtual Java. Ainda, cada implementação de

máquina virtual pode trazer dentro de seu código diferentes compiladores *JIT* e assim impor outros tipos de otimizações para executar o código.

3.5 Consumo de Energia em Dispositivos Android

É possível utilizar o código-fonte de um programa para estimar o consumo de energia em um dispositivo. Uma técnica que relaciona o *bytecode* Java em uma plataforma Android com o consumo de energia é descrita por Hao *et al.* (2012). Nesse artigo, cada instrução de *bytecode* foi executada em dez loops cada um com vinte milhões de chamadas de cada instrução. Conectando o dispositivo com um monitor de energia, foi possível concluir sobre o custo de energia de cada instrução em diferentes granularidades para um tipo de processador. Para chamadas de sistemas que recebem parâmetros, foi utilizada uma média de diferentes chamadas encontradas em aplicações disponíveis para *download*.

Os dispositivos de entrada e saída têm um comportamento de rastro. Esse comportamento faz com que após uma chamada de sistema para leitura ou escrita em uma placa wireless, por exemplo, a mesma não passa imediatamente para um estado de menor consumo de energia. Ao invés disso, mesmo depois que a aplicação é finalizada, o dispositivo permanece em seu estado de consumo máximo por algum tempo. Devido a esse fato, para minimizar o consumo de energia, ao enviar dados para a rede é necessário ser cauteloso e tentar não enviar dados *on-demand* e evitar utilizar a rede caso nenhuma conexão esteja ativa (SHARKEY, 2009). O consumo de energia necessário para enviar pacotes pela rede pode influenciar drasticamente no consumo de energia da aplicação (PROCHKOVA; SINGH; NURMINEN, 2012). Ainda, alguns trabalhos mostram que o uso de cache, em alguns casos, pode aumentar o consumo de energia já que a aplicação terá que realizar uma leitura no disco para saber se o objeto procurado está na memória não volátil local (KAPETANAKIS; PANAGIOTAKIS, 2012).

Para plataforma Android algoritmos recursivos não são uma boa escolha já que esses algoritmos consomem maior quantidade de memória na pilha do processo e consequentemente utilizam mais energia em sua execução. Isso foi mostrado em um trabalho que realizou medições de consumo de energia via software (VIEIRA *et al.*, 2012). Algumas otimizações são possíveis também na troca de mensagens envolvendo dispositivos móveis. A utilização de algoritmos de compressão como *gzip* em mensagens *SOAP* e a utilização de protocolos como *Advanced Message Queuing Protocol* (AMPQ) proporcionam um melhor desempenho ao se tratar do consumo de energia (JOHNSEN; BLOEBAUM; EGGUM, 2015).

Uma técnica que tem o objetivo de relacionar cada linha do código fonte com seu consumo de energia de aplicação é descrita por Li *et al.* (2013). Para que isso seja possível, são realizadas duas etapas. Primeiramente, a aplicação que se deseja medir o consumo de energia é executada e seu rastro de execução ou seja, as partes do código que estão executando a cada momento, é rastreada por um componente de software que leva em consideração mudanças de

contexto de *threads* e de processo. Isso é feito por meio de inserções de chamadas para buscar o tempo do relógio no programa analisado. Ao mesmo tempo, outro componente mede o consumo de energia do dispositivo em um dado momento e grava essa informação. A segunda parte do processo é a análise das amostras de energia e seu relacionamento com a execução do código fonte em um dado momento. Um modelo de regressão e estudo estatístico é aplicado sobre os dados para que seja possível tirar conclusões. Para medir o consumo de energia foi utilizado um dispositivo específico que tem cada componente como *CPU*, *WiFi* e *GPS* conectados a um medidor de energia. Dessa forma, foi possível relacionar cada linha de código fonte com seu consumo de energia, prevendo todos picos no consumo de energia da aplicação.

3.6 Medição de Consumo de Energia por Software

Infelizmente os dispositivos móveis comercializados hoje não trazem consigo nenhum hardware que suporte a medição do consumo de energia internamente (MA *et al.*, 2014). Devido à diversidade de dispositivos de entrada e saída e seu comportamento é dificil descobrir exatamente o quanto um hardware específico (ex. CPU) consumiu de energia em um dado momento. A medição via hardware do consumo de energia do circuito de um dispositivo contabiliza todo o circuito existente no mesmo. Devido a esses fatores, alguns trabalhos propõem técnicas para estimar o quanto uma aplicação consumiu da energia de um dispositivo específico. Em (ZHANG *et al.*, 2010) são descritas duas ferramentas, PowerTutor e PowerBooter, que podem ser utilizadas para estimar o consumo de energia em um dispositivo móvel como um *smartphone* que utiliza o Sistema Operacional Android. PowerBooter é utilizado para criação de um modelo de consumo de energia para um dispositivo específico. Essa ferramenta consegue gerar um modelo de consumo de energia para cada componente como a CPU, LCD, GPS, áudio, Wi-Fi e outros componentes de comunicação. O modelo de energia proposto é baseado em variáveis como por exemplo, brilho do LCD e estas variáveis são utilizadas calculando sua influência sobre a energia total em diversos estados em que o hardware se encontra. Para calcular essa influência, um sensor ou a CPU era mantido em seu estado de maior consumo enquanto os outros em seu estado de menor consumo e assim foi medido a energia consumida no circuito eletrônico. Foi possível concluir ainda, que alguns dispositivos, mesmo semelhantes em sua funcionalidade apresentaram uma grande diferença de consumo de energia e assim, uma outra técnica simples, que utiliza o descarregamento da bateria ao invés de medições via hardware foi proposta para a criação do modelo de energia.

Um estudo sobre requisitos envolvidos na construção de um *profiler* de energia levou a construção de um framework para esse fim, chamado de JouleUnit (WILKE; GOTZ; RICHLY, 2013). O trabalho lista ainda, os principais requisitos esperados para um sistema desse tipo:

Capacidade de Carga: Definição e execução da carga de trabalho.

Suporte para Execução de Carga de Trabalho Para executar uma carga de trabalho em situações específicas, o framework deve ser capaz de expressar essa carga em diferentes granularidades. Isso envolve a invocação de apenas um método com duração de poucos segundos e até cenários durando muitos minutos ou horas.

Construir e Reiniciar Ambiente Específico O framework deve proporcionar a preparação do ambiente e após a execução, deve ser capaz de deixar o ambiente da mesma forma em que ele se encontrava antes da execução.

Logs de Eventos Deve fornecer o suporte a *log* de eventos sincronizados com a execução de carga atribuindo o valor da energia antes e depois da execução.

Capacidade de *Profiling*: Caracterização do dispositivo a ser analisado.

Caracterizar o Consumo de Energia O framework deve fornecer a capacidade de caracterizar o consumo de energia no dispositivo. A quantificação desse consumo deve ser executada periodicamente.

Caliabragem Automática Diferentes dispositivos e medidores de energia trabalham em frequências diferentes. O framework deve ser capaz de ajustar e se adaptar de acordo com essa frequência. Ainda é preciso conhecer o consumo de energia quando o dispositivo está em um estado de suspensão (*idle*).

Suporte ao *Profiling* via Software e Hardware A caracterização do consumo de energia utilizando o monitoramento por hardware geralmente causa menos interferências no sistema, porém, o framework deve classificar o dispositivo utilizando monitores de software também.

Monitoramento interno e externo O monitoramento de software é geralmente realizado utilizando um processo ou código sendo executado no dispositivo enquanto o monitoramento de hardware observa o dispositivo em execução. O framework deve proporcionar tanto o monitoramento externo, quanto o interno.

Fácil Integração com Diferentes Dispositivos Atualmente, os dispositivos apresentam uma heterogeneidade em arquitetura e Sistemas Operacionais. Deve ser possível de utilizar a ferramenta implementando apenas algumas classes do framework.

Monitoramento em Paralelo de Diferentes Dispositivos Ambientes e cenários de testes complexos podem exigir o monitoramento e caracterização do consumo de energia de diversos dispositivos ao mesmo tempo. Assim, é preciso que o framework cumpra também com esse requisito.

Aquisição de Outras Informações de Hardware O consumo de energia por si só não é suficiente em muitos casos. Dessa forma, o framework deve fornecer ao usuário outras informações de hardware como utilização de CPU e memória.

Teste e Coordenação: Esse terceiro requisito envolve a execução de carga de trabalho sincronizado com o monitoramento de energia e eventos do sistema.

Sincronização para a Execução da Carga de Trabalho Para garantir que a energia de toda a carga de trabalho seja contabilizada, o captura de informações deve ser iniciada no momento da execução e finalizada logo que a execução da carga terminar.

Execução de Múltiplos Testes Para que se consiga resultados válidos, muitas repetições devem ser executadas e uma abordagem estatística deve ser utilizada. O framework deve proporcionar facilmente a execução de repetições dos testes.

Sincronização de Eventos Em casos que a medição de energia no sistema é executada periodicamente, esses eventos devem ser sincronizados com os eventos de execução de carga no sistema.

Sincronização do Relógio do Sistema Como os relógios do dispositivo a ser analisado e do computador que está capturando os dados podem estar em horários diferentes, é necessário que ambos sejam sincronizados para que a captura de eventos seja facilitada.

Validação e Apresentação dos Resultados: Esse último grupo de requisitos envolve a validação e representação dos dados obtidos nos experimentos.

Validação de Resultados Quando se realiza experimentos com resultados não determinísticos, a variável de resposta geralmente apresenta uma variância, assim, o framework deve incluir um suporte básico a ferramentas estatísticas.

Exportação de Resultados Para que seja possível analisar os dados obtidos e tirar conclusões sobre o experimento, a ferramenta deve proporcionar a exportação de dados em um formato bem suportado por outros softwares.

Apresentação de Resultados Para que o usuário consiga ver rapidamente o comportamento do sistema, o framework deve incluir funções para visualização dos dados através de gráficos ou tabelas.

Na plataforma Android é possível também estimar o consumo de energia de uma aplicação utilizando as informações fornecidas pelo kernel. Isso é possível pois o *SDK* Android fornece informações de voltagem, nível e temperatura da bateria e o estado de diversos componentes como a sensores, tela e interfaces de comunicação. Ainda, como o sistema é baseado em UNIX, o kernel fornece informações de uso de memória, estado de um processo, uso da CPU, frequência atual da CPU e dados de armazenamento através de um diretório virtual conhecido como *proc filesystem* (CORRAL *et al.*, 2013).

Em um ambiente de grade móvel, é importante que seja considerada não apenas a execução do código, mas também a energia utilizada na comunicação entre os nós. Esses fatores são determinantes para saber se uma aplicação pode ser paralelizada nesse ambiente. Um trabalho que foca no consumo de energia tanto dos dispositivos de entrada e saída quanto do processador,

mostra que estimar o consumo de uma aplicação não é uma tarefa trivial (PATHAK; HU; ZHANG, 2012).

3.7 Wakelocks e Bugs de Energia na plataforma Android

Assim como nos computadores desktop, em um *smartphone* o processador pode estar em diferentes estados de consumo de energia. O estado inicial é aquele em que o *System on Chip* não consome nenhuma energia. Quando o processador é acordado, ele pode estar em um estado de voltagem parcial, essa técnica é conhecida como *Dynamic Voltage/Frequence Scaling* (DVFS) (RUMI; ASADUZZAMAN; HASAN, 2015). O processador *ARM* presente no Google Nexus One suporta 7 diferentes estados de voltagem (JINDAL *et al.*, 2013). Outro fato importante é que o sistema operacional pode desligar e ligar componentes de entrada e saída como GPS, Wifi e outros sensores, quando o mesmo não estiver sendo utilizado.

Ao se tratar da execução de aplicações que demandam processamento por um longo tempo, o desligamento do processador pelo sistema operacional é um grande problema. Para que o programador previna que o sistema operacional não desligue a *CPU*, é possível utilizar uma chamada de sistema para adquirir uma *WakeLock*. Este componente, representado por um objeto Java, garante que o processador continue em seu estado de potência máximo até o processo o liberar. Embora este recurso seja importante no desenvolvimento de aplicações, o mesmo abre possibilidades onde um código mal programado possa provocar um grande consumo de energia no dispositivo.

A implementação de *drivers* em *smartphones* englobam funções responsáveis por ligar ou desligar o componente. Isso abre possibilidades para a existência de novos tipos de *bugs*. Como exemplo, se um dispositivo estiver em seu estado de consumo máximo de energia e o processador estiver desligado, esse componente continuará nesse estado de consumo máximo já que o código necessário para o desligamento do mesmo ficará impossível de ser executado (JINDAL *et al.*, 2013).

3.8 Considerações Finais

Neste capítulo, foram discutidos diversos conceitos importantes para o desenvolvimento do projeto. Esses conceitos serão úteis para o desenvolvimento de uma metodologia para avaliação do consumo de energia em um sistema de grade móvel, a qual será descrita no próximo capítulo.

CAPÍTULO

4

PROPOSTA DE UM *MIDDLEWARE* PARA GRADE MÓVEL

4.1 Considerações Iniciais

Com o objetivo de mostrar que a proposta de execução de aplicações paralelas em grades móveis é válida, este capítulo apresentada primeiramente a proposta de um *middleware* para execução dessas aplicações em uma grade móvel. Como já descrito, alguns problemas são apresentados quando essa abordagem é utilizada. Com isso, utilizando o protótipo do *middleware* proposto, foi realizado um estudo sobre o consumo de energia de uma aplicação que utiliza duas estruturas de dados diferentes. Os resultados obtidos com esse estudo ilustram a importância de se projetar e implementar corretamente algoritmos que são que serão executados em um dispositivo que possua restrição de bateria.

4.2 Arquitetura do *middleware* para grade móvel

Como estudo inicial, foi desenvolvido um protótipo de um *middleware* para a plataforma Android com o intuito de paralelizar um código entre dispositivos conectados em uma mesma rede sem fio. Esse protótipo foi construído utilizando uma arquitetura em camadas onde cada uma fornece uma interface e um serviço para a camada acima. Os serviços implementados por cada camada são:

Camada da Aplicação Android: Essa camada é o ponto de entrada para a aplicação Android. Ela fornece uma interface gráfica para o usuário disparar tarefas (objetos java pré-programados e compilados) pela rede.

Camada Broker/Resource: Essa camada contem as funcionalidades mais importantes para o *middleware*. Os serviços implementados por ela são: (i) recebimento de requisições,

(ii) distribuição de tarefas; (iii) descoberta de recursos; (iv) execução de código; e (v) recuperação de dados.

Camada de Comunicação: Essa camada contém todo o código necessário para comunicação entre *hosts* em uma rede, ela utiliza chamadas de sistema de leitura e escrita em *sockets Transmission Control Protocol* (TCP). Essa camada fornece mecanismos para escutar requisições na rede, envio e recebimento de objetos e estabelecimento de conexões.

Camada do Sistema Operacional Android: É a camada que constitui o Sistema Operacional em si. Essa camada fornece serviços que facilitam a comunicação e coordenação entre os processos.

Aproveitando o recurso de serialização e migração de código da plataforma Java, cada tarefa foi representada por um objeto Java do tipo *GridJob*. Assim um nó não precisava conhecer a implementação de uma tarefa específica, ao invés disso, o nó conhecia apenas a interface *GridJob* e seu método *execute*. Esse método retornava um outro objeto como resposta que era então enviado para o dispositivo que havia feito a requisição.

A Arquitetura de Sistema de um Sistema Distribuído define como os componentes de software estarão organizados em cada nó (TANENBAUM, 2007). Nesse aspecto, o software desenvolvido para esse experimento contava com todos os componentes em cada dispositivo. Assim, qualquer dispositivo poderia submeter novos *jobs* para serem executados. Dessa forma, como mostra a Figura 3, foi possível multiplicar matrizes utilizando três *smartphones* conectados por uma rede sem fio.

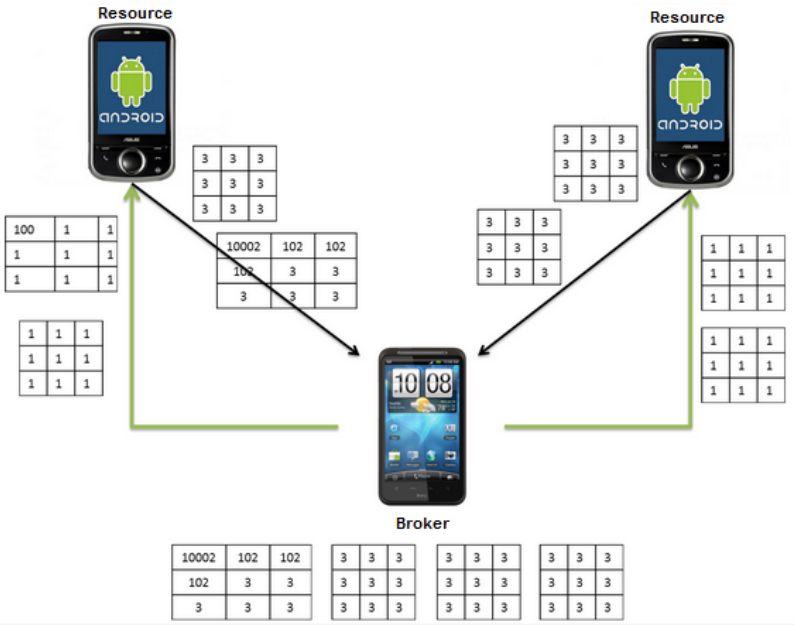

Figura 3 – Multiplicação de Matrizes utilizando um protótipo de *Middleware* Móvel

De acordo com esse experimento, a criação de um *middleware* para essa plataforma é uma prática viável, porém, é preciso adicionar componentes que assegurem a integridade dos

dados, adicionem tolerância a falhas e considerem as restrições de bateria e processamento para que seja possível obter um melhor desempenho.

4.3 Estudo de uma aplicação: multiplicação de matrizes

Uma avaliação de desempenho foi realizada utilizando o *middleware* proposto. Alguns experimentos foram definidos e realizados com o objetivo de descobrir o quanto o projeto e implementação de algoritmos distribuídos podem impactar no desempenho e no consumo de energia em uma grade computacional móvel.

A aplicação escolhida para fazer essa validação foi a de multiplicação de matrizes, a qual é necessária em várias aplicações científicas. Foram desenvolvidos dois algoritmos de multiplicação de matrizes (diferenciados como *Algoritmo A* e *Algoritmo B*) em linguagem de programação Java. Tais algoritmos fazem exatamente a mesma coisa, mas diferença na estrutura de dados utilizada. O *Algoritmo A* utiliza um *array* de matrizes, ou seja, um *array* de três dimensões, enquanto o *Algoritmo B* utiliza três *arrays* de uma dimensão. No segundo caso, cada matriz é representada utilizando os três primeiros índices de cada *array*. As Figuras 4 e 5 ilustram a diferença entre as estruturas de dados utilizadas nos dois algoritmos.

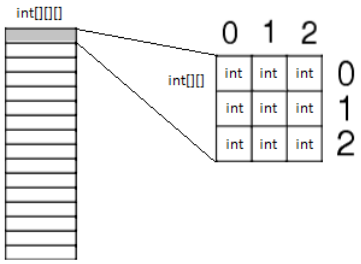

Figura 4 – Estrutura de dados utilizada pelo Algoritmo A

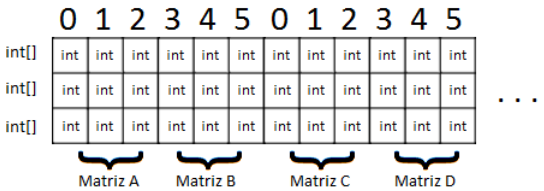

Figura 5 – Estrutura de dados utilizada pelo Algoritmo B

4.3.1 Planejamento da Avaliação de Desempenho

De modo a analisar o desempenho dos dois algoritmos, foi feito um planejamento de experimentos com três fatores: a carga de trabalho, o dispositivo Android e o algoritmo utilizado como *benchmark*. A Tabela 1 indica os fatores e níveis utilizados. As variáveis de Resposta

Fator	Níveis
Carga de Trabalho	70 mil multiplicações de matrizes 50 mil multiplicações de matrizes
Hardware	Galaxy S3 mini – 1GHz Dual Core, 1GB RAM Xperia Arc S – 1.4GHz Single Core, 512MB RAM
Algoritmo	Algoritmo A Algoritmo B

Tabela 1 – Fatores e Níveis envolvidos no experimento.

desse experimento foram: Tempo de execução (em milissegundos), Consumo de Energia (mAh) e Memória utilizada na *heap* da Máquina Virtual Dalvik (em MB).

Com os fatores, níveis e variáveis de resposta definidos, os testes foram executados baseando-se na avaliação de desempenho do tipo fatorial completo, totalizando oito experimentos. Foi definido um intervalo de confiança de 95% para cada experimento, e cada um foi replicado 10 vezes, resultando num total de 80 testes. A Tabela 2 ilustra a configuração de cada experimento e a quantidade de replicações realizadas para cada um.

Carga de Trabalho	Hardware	Algoritmo de Teste	Total de Execuções
70 mil multiplicações de matrizes	Galaxy S3 mini – 1GHz Dual Core, 1GB RAM	Algoritmo A	10
70 mil multiplicações de matrizes	Galaxy S3 mini – 1GHz Dual Core, 1GB RAM	Algoritmo B	10
70 mil multiplicações de matrizes	Xperia Arc S – 1.4GHz, 512MB RAM	Algoritmo A	10
70 mil multiplicações de matrizes	Xperia Arc S – 1.4GHz, 512MB RAM	Algoritmo B	10
50 mil multiplicações de matrizes	Galaxy S3 mini – 1GHz Dual Core, 1GB RAM	Algoritmo A	10
50 mil multiplicações de matrizes	Galaxy S3 mini – 1GHz Dual Core, 1GB RAM	Algoritmo B	10
50 mil multiplicações de matrizes	Xperia Arc S – 1.4GHz, 512MB RAM	Algoritmo A	10
50 mil multiplicações de matrizes	Xperia Arc S – 1.4GHz, 512MB RAM	Algoritmo B	10

Tabela 2 – Experimentos Realizados.

Foi instalada a ferramenta de monitoramento *LittleEye* no computador no qual os dispositivos Android são conectados. O computador apenas serviu como a plataforma para executar o *LittleEye* e a coleta de dados foi realizada diretamente via cabo USB, não havendo necessidade de inicializar ou reinicializar o sistema operacional do computador utilizado durante os testes. A ferramenta *LittleEye* mostra a energia consumida, tanto em seu total (consumo total de energia no aparelho) como a energia consumida apenas pelo processador, rede ou da tela (*display*). Ainda, a ferramenta fornece também a memória utilizada pelo processo no sistema operacional (total), a memória consumida pela *heap* da máquina virtual e o *Proportional Set Size* (PSS), que é um calculo estatístico utilizado pelo Sistema Operacional Android para finalizar algum processo quando o sistema se encontra em uma situação de falta de memória.

4.3.2 Análise dos Resultados

Uma comparação dos resultados considerando-se as variáveis de resposta tempo de resposta, consumo de energia, e memória podem ser observados nas Figuras 6, 7 e 8, respectivamente.

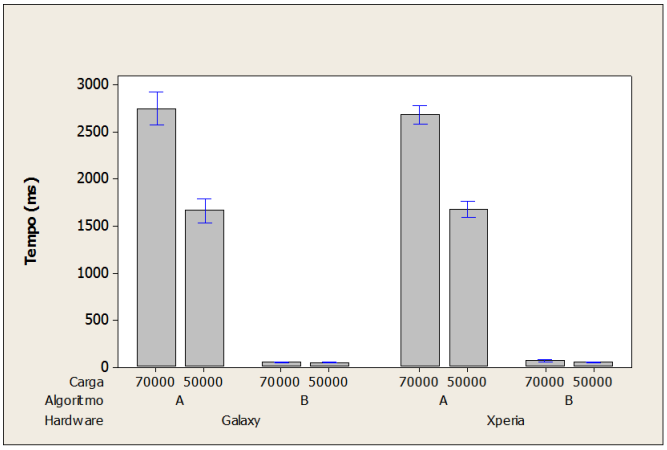

Figura 6 – Comparação entre os tempos de resposta obtidos nos experimentos

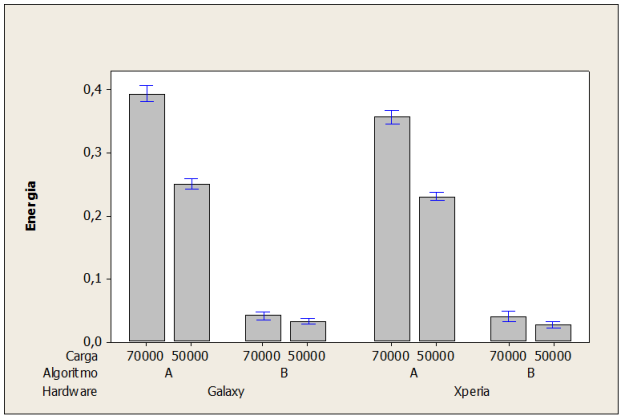

Figura 7 – Comparação entre o consumo de energia nos experimentos

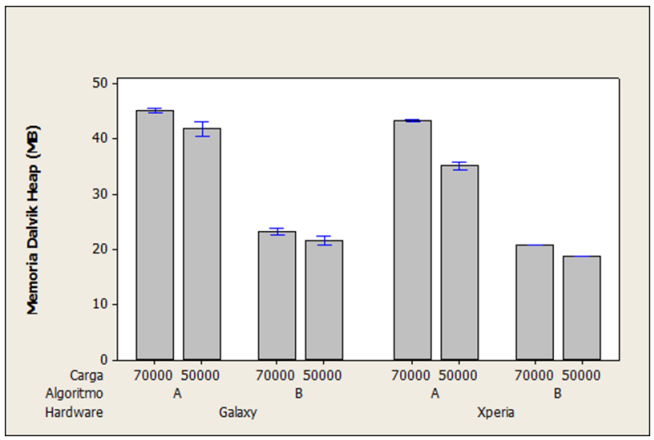

Figura 8 – Comparação entre a memória utilizada nos experimentos

Na Figura 6 observa-se que existe uma grande diferença no tempo de execução da aplicação considerando-se o *Algoritmo A* e o *Algoritmo B*. O mesmo não acontece entre os diferentes dispositivos pois, como a aplicação utilizava apenas uma *thread*, o núcleo extra do *smartphone Galaxy S3* não influenciou na variável de resposta. Ainda, o consumo de energia (Figura 7) também foi menor para o *Algoritmo B* devido ao fato de que esse algoritmo executou

bem mais rápido e assim ficou menos tempo processando.

É importante ressaltar também a diferença entre o gasto de memória entre os dois algoritmos (Figura 8). É possível concluir que *arrays* multidimensionais consomem maior quantidade de memória nessa plataforma. Este fato pode ser melhor compreendido analisando o *bytecode* gerado no acesso a essas estruturas de dados. O código listado a seguir foi utilizado para gerar o código que está logo em seguida. Para isso foi utilizada a ferramenta disponível no *Software Development Kit* (SDK) Java chamada *javap*. A definição das instruções utilizadas nas listagens pode ser observada na Tabela 3.

```
1  public void acess(int[] array) {
2    int value = array[3];
3  }
4
5  public void acessMulti(int[][][] array) {
6    int value = array[1][2][3];
7  }
```

```
1  public void acess(int[]);
2    Code:
3       0: aload_1
4       1: iconst_3
5       2: iaload
6       3: istore_2
7       4: return
8
9  public void acessMulti(int[][][]);
10   Code:
11      0: aload_1
12      1: iconst_1
13      2: aaload
14      3: iconst_2
15      4: aaload
16      5: iconst_3
17      6: iaload
18      7: istore_2
19      8: return
```

É possível observar que o acesso a um *array* multidimensional (public void acessMulti) requer uma maior quantidade de acessos a memória, pois essa estrutura de dados é representada

Instrução	Descrição
aload_1	Carrega uma referência para a pilha de uma variável local
iconst_X	Carrega o valor X para a pilha do programa
aaload	Carrega a referência de um vetor para a pilha
iaload	Carrega um inteiro de um array
istore_X	Carrega o valor para a variável int declarada como value no código fonte

Tabela 3 – Instruções de bytecode Java utilizadas para acessar arrays (AMERICA, 2007).

na linguagem java por um *array* de *arrays*. Ainda é possível concluir que para alocar a memória nos *arrays* multidimensionais, é preciso executar diversas operações *new*, enquanto no *array* unidimensional é feita apenas uma chamada para alocação de memória.

Com os gráficos de influência de fatores do experimento realizado foi possível compreender o quanto a implementação de um algoritmo foi importante nesse experimento.

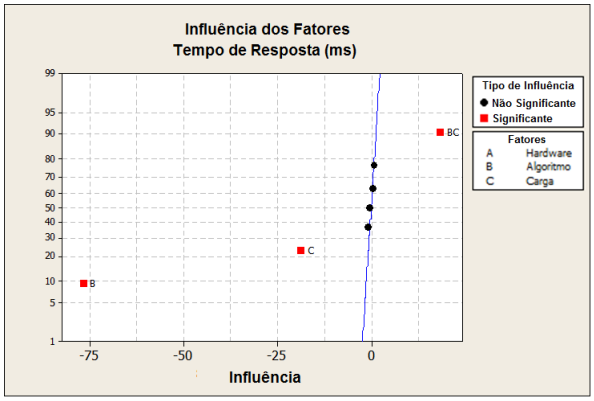

Figura 9 – Influência dos fatores para a variável de resposta Tempo

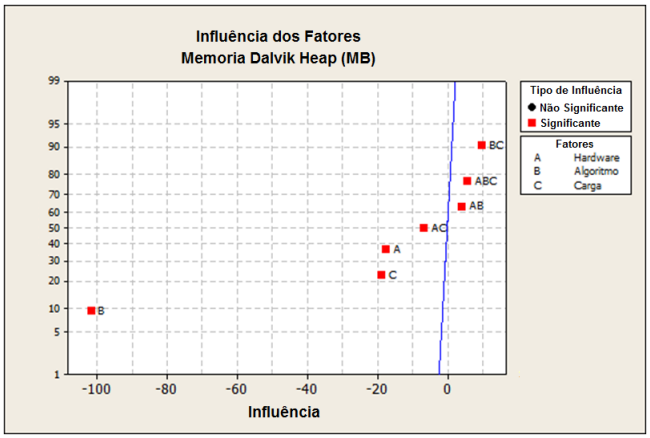

Figura 10 – Influência dos fatores para a variável de resposta Memória

Analisando a variável de resposta tempo (Figura 9), é possível concluir que para esse experimento o fator que mais influenciou foi o algoritmo, seguido pelo fator carga. Ainda, a combinação dos dois fatores também influenciou essa variável. A carga de trabalho poderia apresentar uma maior influência caso os níveis escolhidos fossem valores mais distantes. Porém

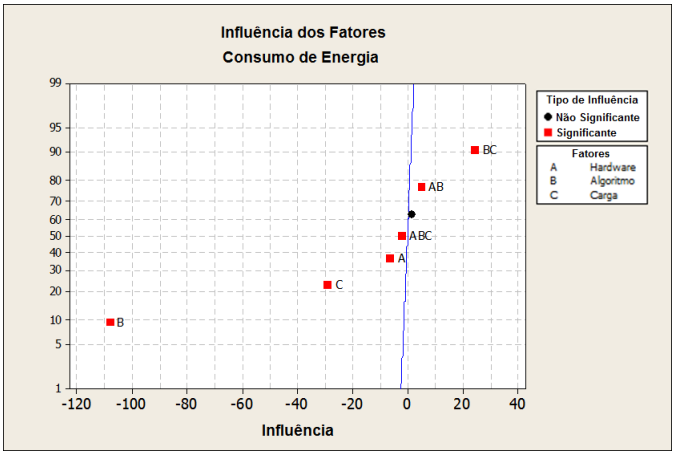

Figura 11 – Influência dos fatores para a variável de resposta Energia

isso não foi possível, já que a memória utilizada pelo *Algoritmo A* consumia uma enorme quantidade e causava o erro de *OutOfMemoryException* no dispositivo *Xperia* que contava com apenas 512MB. Sobre o fator hardware é possível ver que o dispositivo *Xperia* conta com um processador com maior *clock* porém com apenas um núcleo de processamento processamento. Como a aplicação utilizou, para realizar o processamento dos algoritmos, apenas uma *thread* o fator hardware não influenciou na variável de resposta Tempo mesmo com a diferença de número de núcleos e velocidade do *clock* dos processadores.

A memória medida como variável de resposta foi a memória utilizada pela *heap* da Máquina Virtual Dalvik. Essa memória é a memória utilizada apenas pelo processo que está sendo executado. Como na plataforma Android, que é um sistema Linux, todo processo é criado a partir de uma chamada *fork* em um processo pai, chamado de Zygote, os processos podem compartilhar páginas de memória contendo código de bibliotecas e o próprio código da máquina virtual. Dessa forma, a memória utilizada pelo processo pode ser calculada somando a memória privada mais a memória compartilhada entre os processos[1]. A Figura 10 apresenta o gráfico de influência de fatores da variável de resposta memória, onde é possível ver que, embora o algoritmo e a carga de trabalho sejam os fatores que mais influenciaram na variável, o hardware apresentou uma pequena influência. Esse fato pode ser explicado pois os Sistemas Operacionais e versões da Maquina Virtual Dalvik eram diferentes nos dispositivos, assim diferentes versões de bibliotecas e código foram carregados na memória.

Em relação a variável de resposta energia (Figura 11), como a aplicação executada foi uma aplicação totalmente *CPU bound*, o fator que mais influenciou esta variável de resposta também foi o algoritmo, seguido da carga de trabalho e do hardware. É possível ver no gráfico que existe uma influência negativa na variável energia para o dispositivo *Xperia*. Esse hardware, por ser menos potente e contar com apenas um núcleo consumiu uma menor quantidade de energia na execução dos *benchmarks*. Existe ainda, uma pequena interação entre os três fatores

1 <http://developer.android.com/tools/debugging/debugging-memory.html>

sobre essa variável de resposta.

Uma questão importante neste experimento é que, como na linguagem de programação Java, um *array* multidimensional é na verdade um *array* de *arrays*, é impossível alocar memória em apenas uma chamada. Em vez disso, é necessário alocar memória para um *array* e depois realizar muitas chamadas para alocar a memória dos outros *arrays*. Isolando apenas essas chamadas no experimento foi possível observar também que é mais eficiente utilizar apenas uma chamada para instanciar os objetos ou estruturas de dados do que realizar muitas chamadas mesmo que no total a memória consumida seja a mesma.

4.4 Considerações Finais

Esse Capítulo apresentou uma proposta de *Middleware* para computação em grade móvel, ilustrando sua utilização com uma aplicação que pode ser paralelizada. A aplicação foi utilizada para mostrar que a grade móvel pode ser utilizada para processamento paralelo e também mostrou a importância do projeto e implementação de um algoritmo nesse ambiente. Devido a restrição de bateria encontrada nesses dispositivos, um escalonador de tarefas precisa de informações sobre o quanto um dispositivo móvel irá consumir de energia para executar essa tarefa. Para que isso seja possível, é necessário definir uma maneira de estimar esse consumo, o que será apresentado no próximo Capítulo.

CAPÍTULO

5

ESTIMATIVA DE CONSUMO DE ENERGIA: MODELO E VALIDAÇÃO

5.1 Considerações Iniciais

Este capítulo apresenta a metodologia proposta para a estimativa de consumo de energia de aplicações escritas na linguagem Java para dispositivos móveis que utilizam o sistema Android. Como parte da metodologia, são apresentadas a definição e organização dos experimentos utilizados na criação de um modelo responsável por representar o consumo de energia de um dispositivo. Em seguida, são apresentados o ambiente utilizado na execução dos *benchmarks* propostos, os dados coletados com a execução dos *benchmarks* e as funções matemáticas que moldam o consumo de energia dos dispositivos para as operações citadas neste capítulo. Finalizando o capítulo, é apresentada a validação do modelo proposto.

5.2 Criação de um Modelo de Consumo de Energia para um Dispositivo Móvel

Como ilustrado na Seção 4.2, a construção de uma ambiente de grade móvel utilizando *smartphones* para execução de aplicações paralelas é viável, porém existem vários desafios. Um desafio apresentado refere-se ao projeto e implementação dos algoritmos. Outro desafio refere-se ao fato de que nesse ambiente, onde alguns provedores de recursos impõem limites em relação a quantidade de energia a ser utilizada, é preciso estimar o consumo de energia de uma aplicação. Para que isso seja possível, é necessário compreender como a execução de diferentes programas pode influenciar no descarregamento da bateria de um dispositivo.

A metodologia proposta neste trabalho difere-se de outras citadas anteriormente por tratar este problema utilizando uma abordagem em alto nível. Diferente de outras propostas que

investigaram o consumo de energia em relação a instruções de máquina gerados pelo código (HAO *et al.*, 2012) ou o tempo que o processo utiliza o processador (DARGIE, 2015), é proposta uma metodologia para estimar o consumo de energia de uma aplicação utilizando dados que relacionam um dispositivo móvel específico e as operações de cálculos matemáticos contidas no código fonte. A Figura 12 ilustra uma tarefa que precisa de ser executada por um dispositivo móvel em uma grade computacional. Cada um dos dispositivos conta com uma quantidade de bateria restante diferente e o escalonador precisa saber qual a melhor opção para atribuir essa execução. Aparentemente, o *Dispositivo A* parece ser a melhor opção, porém, dependendo da arquitetura desse aparelho, a execução pode precisar de uma quantidade de bateria maior do que o recurso pode proporcionar no momento. Ainda, o *Dispositivo C* pode contar com um hardware específico internamente que o tornaria capaz de executar essa tarefa facilmente.

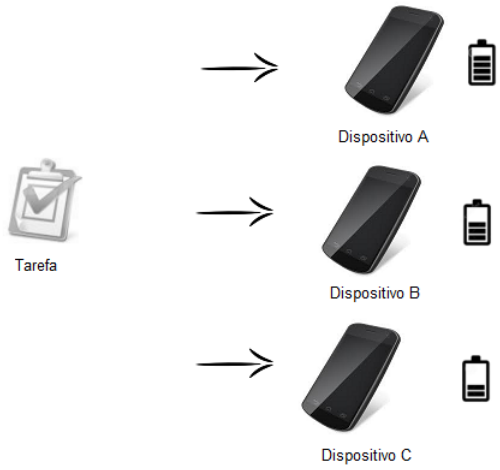

Figura 12 – Tarefa sendo escalonada para um recurso na grade

A situação ilustrada na Figura 12 enfatiza que o escalonador de uma grade computacional móvel precisa conhecer as características do dispositivo e da tarefa a ser executada. Neste trabalho essas características que representam o quanto um dispositivo móvel consome de energia ao executar um determinado código são chamadas de modelo do dispositivo ou *profile*. Utilizando um *profile*, é possível saber quantos *joules* um *tablet* irá consumir de energia ao executar uma determinada carga de trabalho contendo milhões de cálculos matemáticos por exemplo. Um *Diagrama Entidade Relacionamento* (DER) de um banco de dados proposto para manter os *profiles* de cada dispositivo para um sistema escalonador de tarefas é apresentado na Figura 13. Utilizando este banco de dados é possível que novos modelos sejam cadastrados a qualquer momento, adicionando suporte a estimativa do consumo de energia para novos nós provedores de recursos de uma grade computacional móvel.

De acordo com o diagrama é possível ver que cada dispositivo pode ser moldado por mais de um tipo de *Modelo CPU* já que diferentes abordagens podem ser utilizadas ao criar uma representação do consumo de energia do mesmo. Cada *Modelo de CPU* está relacionado a diversos *Comportamentos* que representam qual a formula matemática usada para obtenção de *n*

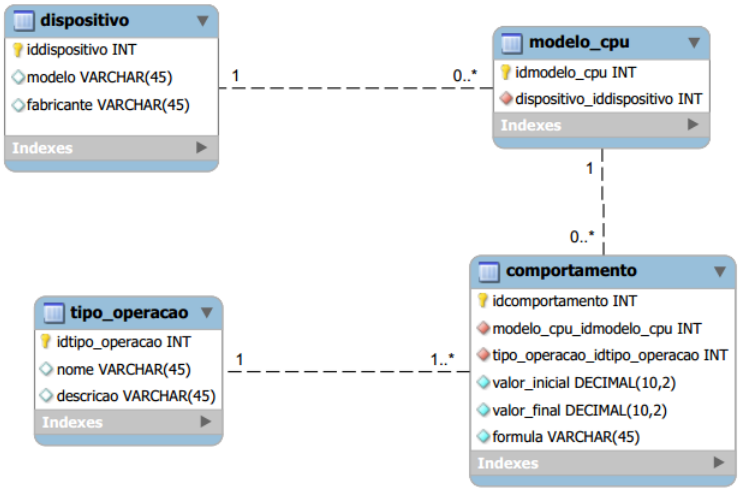

Figura 13 – Diagrama Entidade Relacionamento proposto para armazenar os modelos dos dispositivos móveis

execuções de uma dada operação, sendo *n* um valor entre *valor inicial* e *valor final*. Finalmente cada operação tem que ser cadastrada para que seja possível mapear o *benchmark* em um conjunto de *comportamentos*.

5.3 Definição e Implementação dos *Benchmarks* utilizados nos Experimentos

Como o foco deste trabalho é o de investigar aplicações limitadas por processamento muito utilizadas em aplicações científicas, as chamadas do sistema que envolvem entrada e saída realizadas pela aplicação não fizeram parte dos *benchmarks* criados nos experimentos. Utilizando apenas as operações que envolvem cálculos matemáticos e acesso a memória, foi possível levantar um conjunto de *benchmarks* para serem executados em uma dispositivo móvel.

Esses *benchmarks* foram implementados dentro de uma aplicação Android onde cada uma continha operações comumente encontradas em uma aplicação paralela no contexto de grade computacional. Para entender o comportamento do dispositivo diante de diferentes cargas de trabalho, os testes envolveram operações em diferentes granularidades. Foram implementados algoritmos envolvendo soma, subtração, multiplicação e divisão com diferentes tipos dados, sendo eles, inteiros e números com ponto flutuante (*float* e *double*). Operações lógicas que envolvem comparações entre variáveis como condições *if* não foram consideradas como *benchmarks* pois essas operações não influenciaram as variáveis de resposta em comparação as operações que envolviam cálculos matemáticos.

Os *bechmarks* executados são listados a seguir:

- Soma de Inteiros utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões e quinhentos milhões.

- Soma de Float utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões e quinhentos milhões.
- Soma de Double utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões, quinhentos milhões, um bilhão, um bilhão e duzentos milhões, um bilhão e quinhentos milhões, um bilhão e setecentos milhões e dois bilhões.
- Subtração de Inteiros utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões e quinhentos milhões.
- Subtração de Float utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões e quinhentos milhões.
- Subtração de Double utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões, quinhentos milhões, um bilhão, um bilhão e duzentos milhões, um bilhão e quinhentos milhões, um bilhão e setecentos milhões e dois bilhões.
- Multiplicação de Inteiros utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões e quinhentos milhões.
- Multiplicação de Float utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões e quinhentos milhões.
- Multiplicação de Double utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões, quinhentos milhões, um bilhão, um bilhão e duzentos milhões, um bilhão e quinhentos milhões, um bilhão e setecentos milhões e dois bilhões.
- Divisão de Inteiros utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões e quinhentos milhões.
- Divisão de Float utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões e quinhentos milhões.
- Divisão de Double utilizando diferentes números de operações: um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões, quinhentos milhões, um bilhão, um bilhão e duzentos milhões, um bilhão e quinhentos milhões, um bilhão e setecentos milhões e dois bilhões.

Para os dados do tipo *Double* foi necessário realizar mais experimentos já que esse tipo de dado é o mais utilizado em códigos que envolvem computação científica, obtendo assim uma precisão maior na modelagem.

5.4 Métricas Utilizadas neste Trabalho

É possível analisar as variáveis de resposta e os fatores que influenciam cada uma dessas variáveis de diversas maneiras. Assim, é preciso a descrição de todas as variáveis coletadas e calculadas, bem como os valores que foram adotados em cada fator envolvido nos experimentos.

5.4.1 Potência do Processador

O consumo de energia, ou o trabalho envolvido em um determinado intervalo de tempo *dt* é definido pela seguinte equação:

$$\int_0^t Pdt$$

Esse consumo de energia pode ser representado como a integral da potência por tempo e obtido através da área sobre um gráfico envolvendo essas duas variáveis. Dessa forma, para calcular o consumo de energia de um trabalho realizado pelo processador foi preciso primeiro, obter a potência desse componente quando o mesmo estava executando um código e o tempo total de execução do programa ou trecho deste código analisado.

Os processadores analisados neste trabalho são do tipo *ARM* onde potência e a voltagem desses dispositivos podem ser dinamicamente ajustadas para otimizar seu consumo de energia (KHAN; BILAVARN; BELLEUDY, 2012). O primeiro processador, chamado de *Processador A*, utilizado para executar os experimentos foi o *ARM Cortex-A9 MPCore* em sua versão com dois núcleos e *clock* de 1 Ghz. O segundo dispositivo, chamado de *Processador B*, foi um modelo um pouco mais antigo que utiliza um processador *MSM7227 Cortex-A5* com apenas um núcleo de *clock* 1 Ghz como mostra a tabela 4.

Dispositivo	Modelo	Clock	Núcleos	Smartphone
A	ARM Cortex-A9 MPCore	1 Ghz	2	Samsung Galaxy S III Mini
B	MSM7227 Cortex-A5	1 Ghz	1	Sony Xperia

Tabela 4 – Dispositivos utilizados nos Experimentos

Para obter a potência dos experimentos foram utilizados dois softwares diferentes. O primeiro é um software desenvolvido pela empresa *LittleEye* utilizado também por outras empresas como a montadora de *chipsets Qualcomm*. O software conhece a arquitetura do dispositivo móvel, para os modelos suportados, e utiliza informações do *kernel* do sistema Android para obter a potência do dispositivo em um dado momento. Para validar essas informações outro monitor de potência chamado de *PowerTutor* (ZHANG *et al.*, 2010) foi utilizado. Esse aplicativo foi desenvolvido na Universidade de Michigan por estudantes de Doutorado e ambos os sistemas testados mostraram dados compatíveis sobre a potência do processador.

A Figura 14 mostra a medição de potência utilizando o software *LittleEye*. Nesse caso, a aplicação é executada em um *smartphone* que está conectado no computador ao qual o sistema

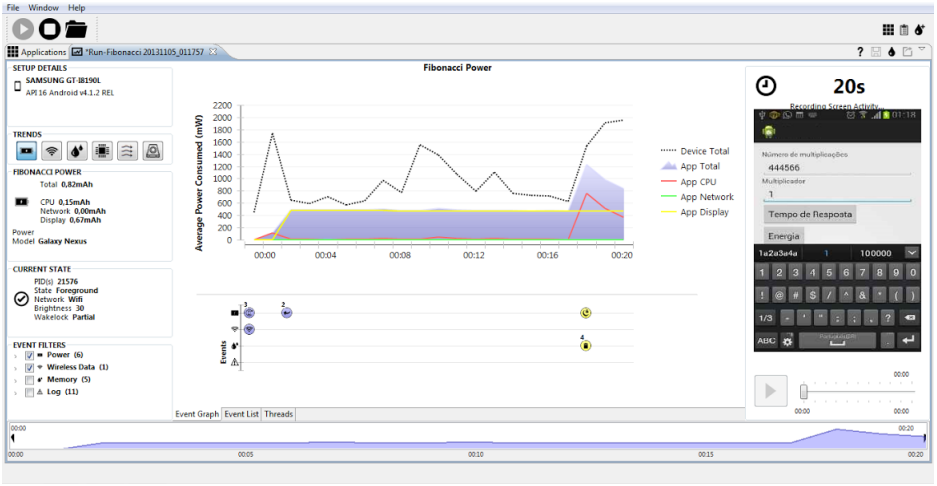

Figura 14 – Medição da potência do processador usando o aplicativo *LittleEye*

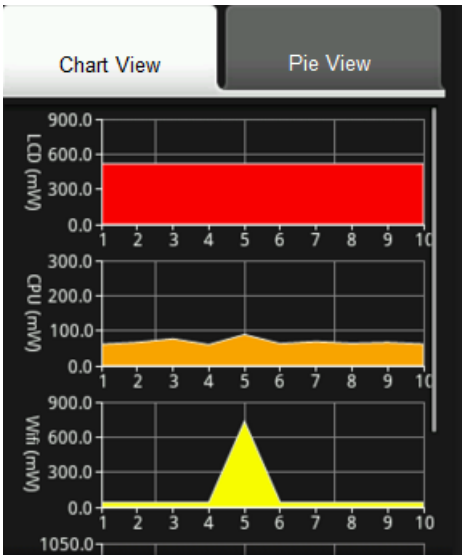

Figura 15 – Medição da potência do processador usando o aplicativo *PowerTutor*

executa e informa os valores obtidos ao longo do tempo. A aplicação que está sendo executada é mostrada à direita na figura e a potência do processador e outros componentes no gráfico central. O outro monitor de potência chamado *PowerTutor*, que também foi utilizado é mostrado na Figura 15. Este aplicativo apresenta um gráfico de potência separado para cada componente. Na imagem é possível observar o comportamento do *LCD*, *CPU* e *Wifi*.

5.4.2 Tempo de Resposta

Além da potência do dispositivo foi preciso calcular o tempo de resposta da execução em cada caso. Esse tempo de resposta é o tempo que o dispositivo levou para processar cada algoritmo com as operações citadas anteriormente. Para obter esses dados antes e depois de cada execução foi colocada uma chamada de sistema para obter o tempo no relógio do dispositivo. Assim, fazendo a diferença entre o valor do tempo depois da execução e o valor do tempo do

relógio antes da execução foi possível obter o tempo total que o aparelho levou para executar cada *benchmark*. Nesse caso, não foi considerado o tempo de escalonamento de outros processos sendo executados no mesmo momento pois esse tempo era desprezível se comparado ao tempo de execução total de cada *benchmark*.

Para cada execução, tanto o tempo de resposta quanto a potência do dispositivo foram calculadas 15 vezes, ou seja, cada *Benchmark* foi executado 15 vezes. Esse número de repetições foi necessário para obter um intervalo de confiança e desvio padrão adequados. Para cada variável foi calculado também a média e o desvio padrão.

5.5 Análise dos Dados e Regressão

Para que fosse possível obter a equação que irá calcular a estimativa do consumo de energia de uma determinada aplicação, foi necessário primeiramente calcular o consumo de energia de cada execução do *benchmark*. O calculo da energia foi obtido através da multiplicação da potência média do processador e do tempo de resposta de cada execução. Para cada operação envolvendo um tipo de dado específico, como por exemplo, multiplicações de variáveis *double*, foram executados testes com diferentes quantidades de multiplicações (um milhão, cinco milhões, dez milhões, vinte milhões, cem milhões, duzentos milhões, quinhentos milhões, um bilhão, um bilhão e duzentos milhões, um bilhão e quinhentos milhões, um bilhão e setecentos milhões e dois bilhões). O gráfico da Figura 16 apresenta as médias para o *benchmark* de multiplicações de variáveis *double*. Ao analisar este gráfico, pode-se ter uma impressão errada sobre o consumo de energia, que aparenta ter um comportamento linear sempre que o número de operações é aumentado. Porém, é possível perceber que isso não acontece para pontos próximos a origem do gráfico, os quais parecem formar uma curva. A Figura 17 apresenta os resultados de forma mais clara dividindo o gráfico em duas partes. Assim, é possível observar que os pontos próximos a origem, na verdade, apresentam um comportamento quadrático enquanto os outros pontos aumentam de forma linear.

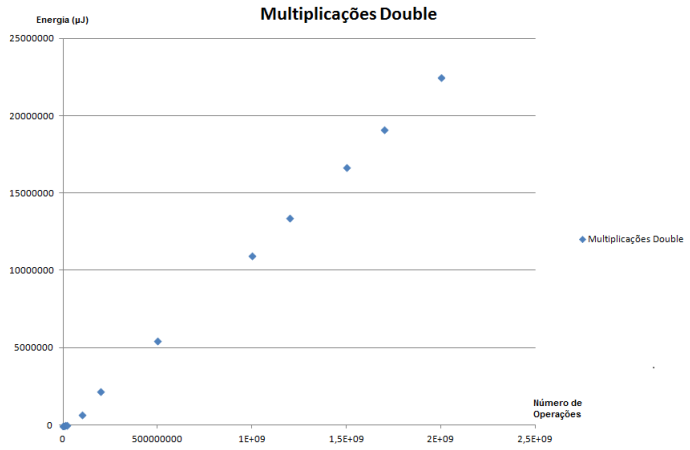

Figura 16 – Médias para as multiplicações de *double* total

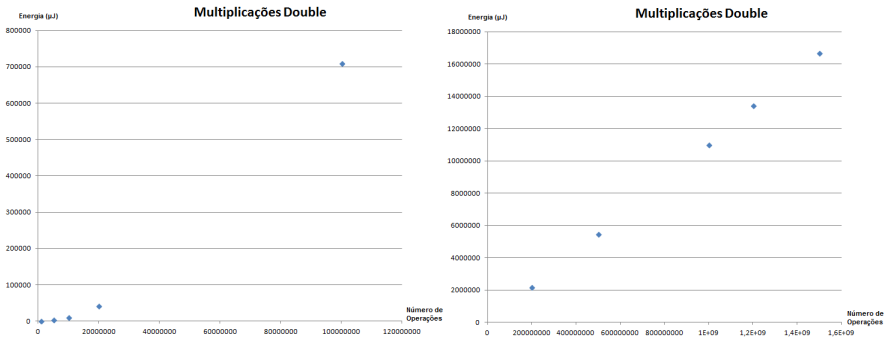

Figura 17 – Médias para as multiplicações de *double* em diferentes intervalos

Observando os gráficos de potência e tempo de resposta separadamente nas Figuras 18 e 19, é possível observar que o motivo do consumo de energia se comportar diferente nas situações de carga alta e carga baixa é a potência medida. Quando o número de operações executadas tem um valor baixo, a potência do processador aumenta de acordo com a carga de trabalho. Por outro lado, quando os experimentos são executados em carga alta, o processador chega em sua potência máxima e essa potência fica constante mesmo com o aumento da carga de trabalho. Nesse caso, apenas o tempo de resposta passa a aumentar.

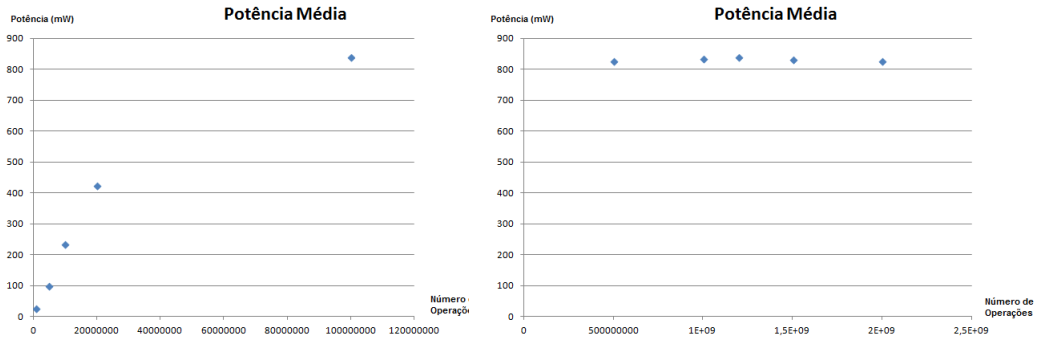

Figura 18 – Variação da potência de acordo com a carga de trabalho

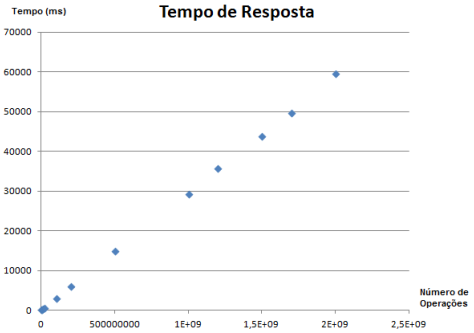

Figura 19 – Variação do tempo de resposta de acordo com a carga de trabalho

Para obter uma função representativa do consumo de energia do dispositivo foi necessário utilizar um método de ajuste de curva. Essa técnica de regressão tenta encontrar uma equação que se ajuste aos pontos de um gráfico. Para todos os *benchmarks* citados anteriormente o consumo

de energia foi moldado utilizando-se de duas equações diferentes: uma para a parte quadrática e outra para a parte linear.

Considerando-se a modelagem do banco de dados apresentada na Figura 13, cada *valor inicial* e *valor final* de um *comportamento* foi definido para suportar intervalos com equações diferentes.

Para todas as operações listadas anteriormente, foram calculados o tempo de resposta e a potência média. Logo em seguida foi obtido o consumo de energia multiplicando as duas variáveis.

A seguir, serão apresentadas as relações encontradas entre o número de operações executadas e o consumo de energia, assim como a diferença no consumo de energia e tempo de resposta entre as diferentes operações.

5.6 Consumo de energia quando o dispositivo não está em seu estado de potência máxima

A Figura 20 mostra os dados obtidos para operações de soma de variáveis de tipos *inteiro*, *float* e *double* para o *Dispositivo A*. Mostra também a função obtida através de regressão utilizando o Método dos Quadrados Mínimos para esses casos.

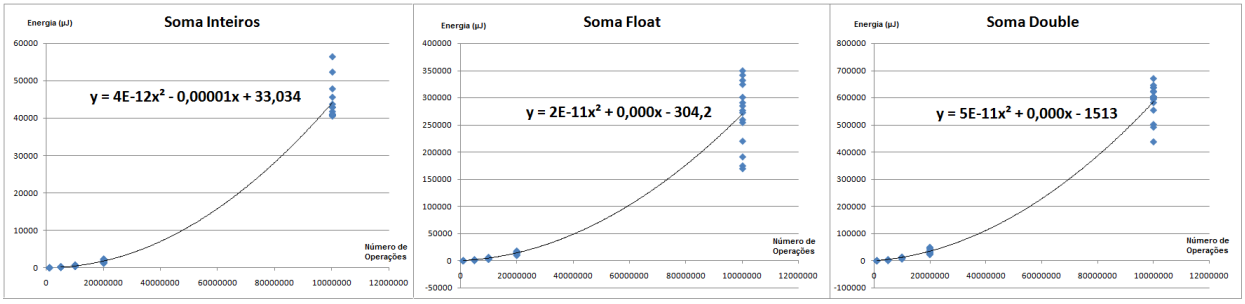

Figura 20 – Operações de soma para o *Dispositivo A*

Cada ponto neste gráfico relaciona o número de operações e o consumo de energia para uma execução do *benchmark*. Analisando o gráfico, é possível notar que a potência do hardware e o tempo de resposta aumentam de forma linear cada um. Devido a esse fato, o consumo de energia, calculado como o produto das duas variáveis, aumenta de maneira quadrática.

Os experimentos envolvendo subtração executados no *Dispositivo A* são mostrados na Figura 21. Pode-se observar que o consumo de energia das operações de soma e subtração apresentam resultados próximos e que para esses casos o tipo de dado com maior custo é o *double*. O motivo disso é que esse tipo de variável é representada utilizado 64 bits na plataforma Android enquanto que variável *float* utiliza apenas 32 bits.

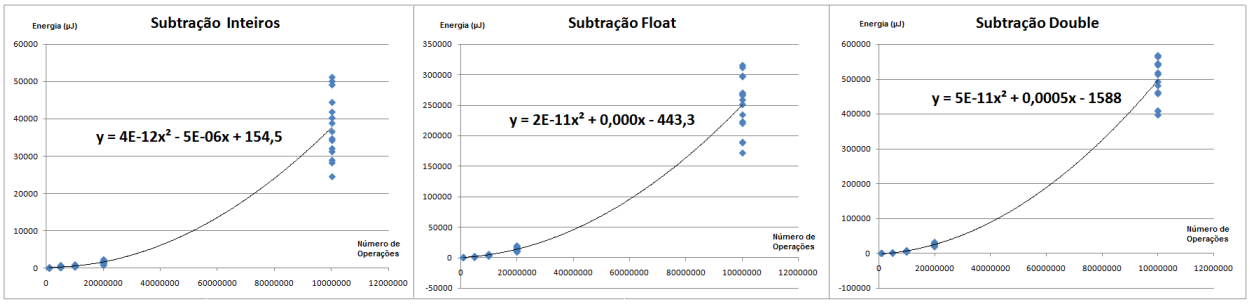

Figura 21 – Operações de subtração para o *Dispositivo A*

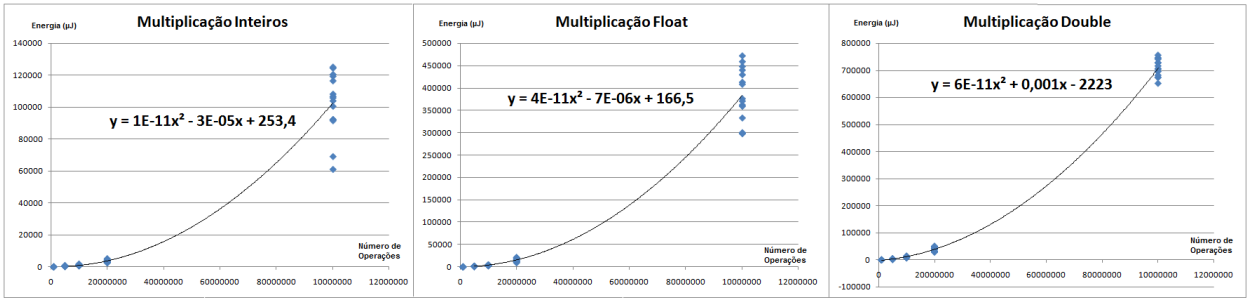

Figura 22 – Operações de multiplicação para o *Dispositivo A*

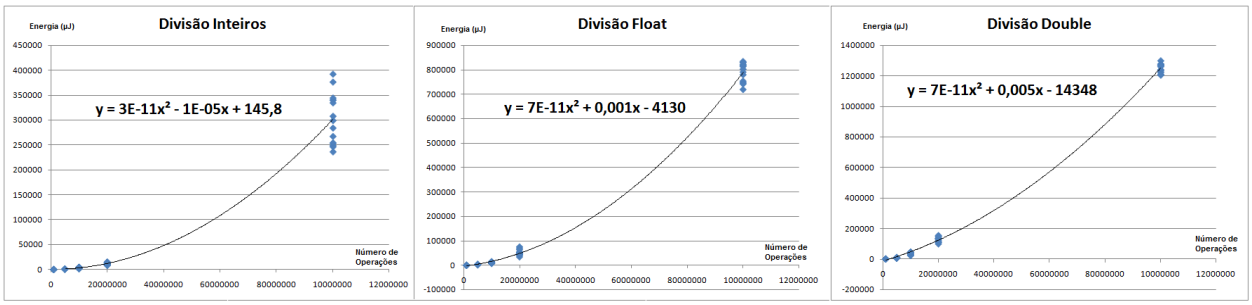

Figura 23 – Operações de divisão para o *Dispositivo A*

As Figuras 22 e 23 apresentam as equações e os gráficos para as operações de multiplicação e divisão executadas no *Dispositivo A* respectivamente. Para esses casos, as operações envolvendo variáveis *double* obtiveram uma menor variância e um maior tempo de resposta e consumo de energia.

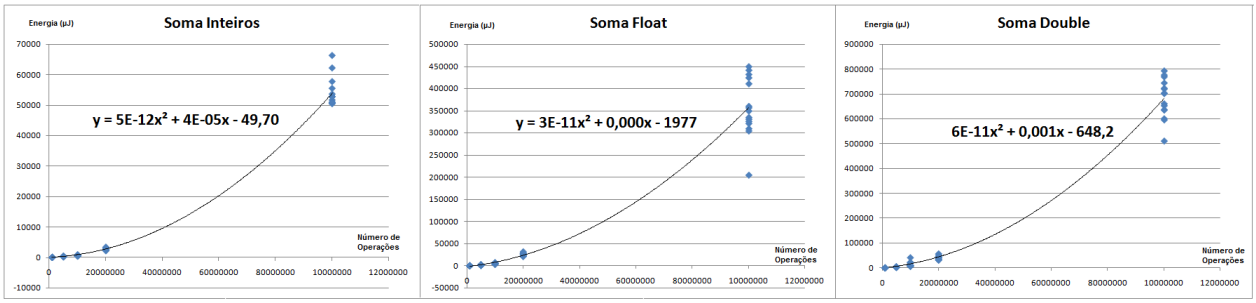

Figura 24 – Operações de soma para o *Dispositivo B*

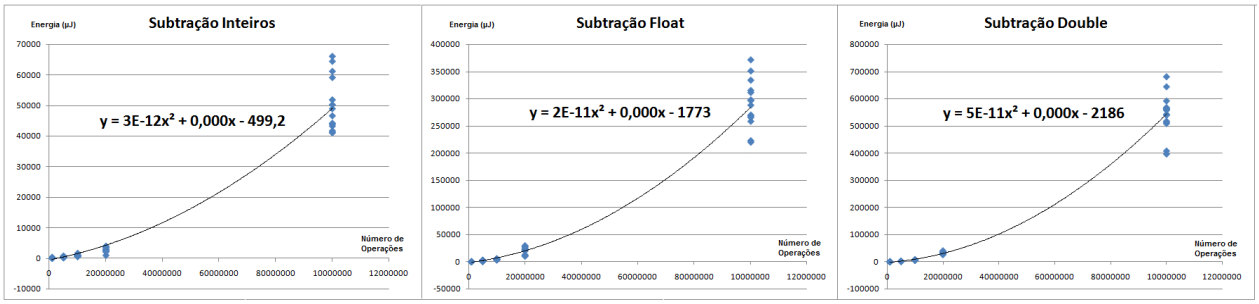

Figura 25 – Operações de subtração para o *Dispositivo B*

Os mesmos experimentos foram executados no *Dispositivo B* e os resultados desses experimentos são mostrados nos gráficos das Figuras 24, 25, 26 e 27. Observando as equações obtidas é possível perceber que o *Dispositivo B* consumiu uma maior quantidade de energia já que o mesmo é um aparelho mais antigo.

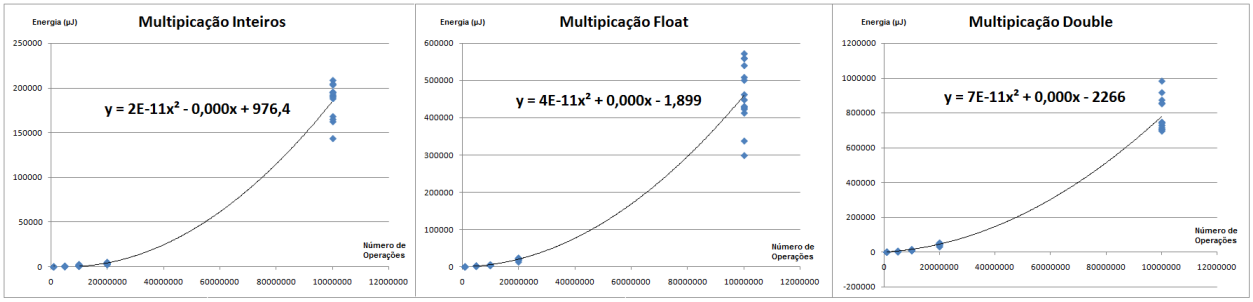

Figura 26 – Operações de multiplicação para o *Dispositivo B*

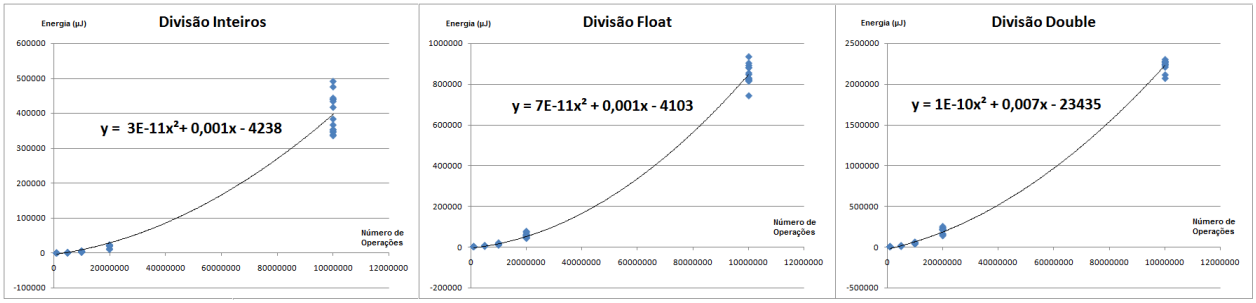

Figura 27 – Operações de divisão para o *Dispositivo B*

Observando os gráficos e as equações obtidas para ambos os *smartphones* é possível observar que os experimentos de subtração apresentaram valores próximos aos experimentos envolvendo soma para variáveis do tipo *double*. A operação de multiplicação gasta mais energia que as operações de soma e subtração e a operação de divisão é a mais energeticamente cara, gastando praticamente o dobro de energia do que gasta a operação de soma. Essa comparação fica mais clara com o gráfico da Figura 28 que também mostra essa relação.

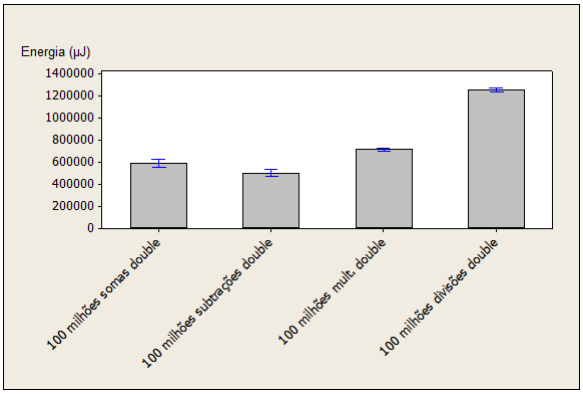

Figura 28 – Comparação entre o consumo de eneregia de cada operação para o *Dispositivo A*

5.7 Consumo de energia quando o dispositivo está em seu estado de potência máxima

Ao aumentar o número de operações executadas nos experimentos foi notado que a potência do processador se tornava constante a partir de um certo ponto. Isso aconteceu, pois o processador estava utilizando sua potência máxima. Para o *Dispositivo A* esse valor era de *840 milliwatts* enquanto o *Dispositivo B* chegou até *1260 milliwatts*. Como o consumo de energia é uma grandeza linearmente proporcional ao tempo de resposta e a potência do hardware, esse consumo passou a aumentar linearmente ao aumentar a carga de trabalho. Dessa forma, é possível representar o consumo de energia através da equação de uma reta.

As Figuras 29 e 30 mostram os gráficos e as equações obtidas na regressão para os casos de operações executadas com variáveis do tipo double para o dispositivo *A*.

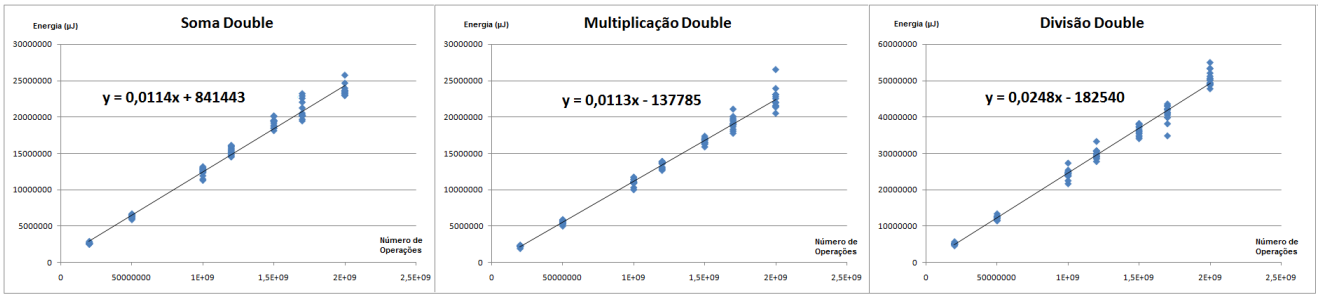

Figura 29 – *Dispositivo A* em sua potência máxima

Observando a comparação entre cada tipo de operação (soma, subtração, multiplicação e divisão) é possível concluir que a operação de divisão é a mais onerosa de todas. É possível observar esse fato quando o dispositivo está variando sua potência, mostrado na Figura 28 e também quando o dispositivo está em seu estado de potência máxima como mostrado na Figura 31.

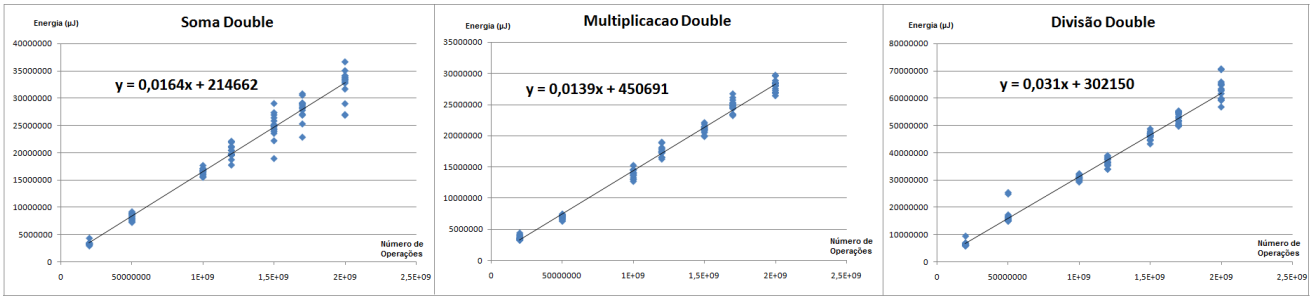

Figura 30 – *Dispositivo B* em sua potência máxima

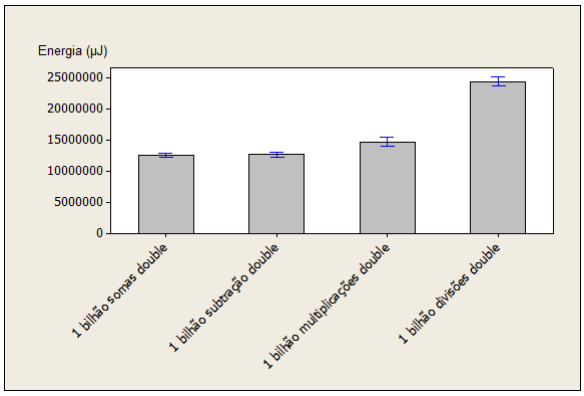

Figura 31 – Comparação entre o consumo de energia de cada operação para o *Dispositivo A*

5.8 Estudo de Caso

As funções obtidas com os *benchmarks* representam o consumo de energia de um dispositivo móvel quando submetido a uma dada carga de trabalho. Dessa forma, ao analisar o código de uma aplicação contendo as operações analisadas é possível estimar seu consumo de energia total, somando o custo de cada uma individualmente.

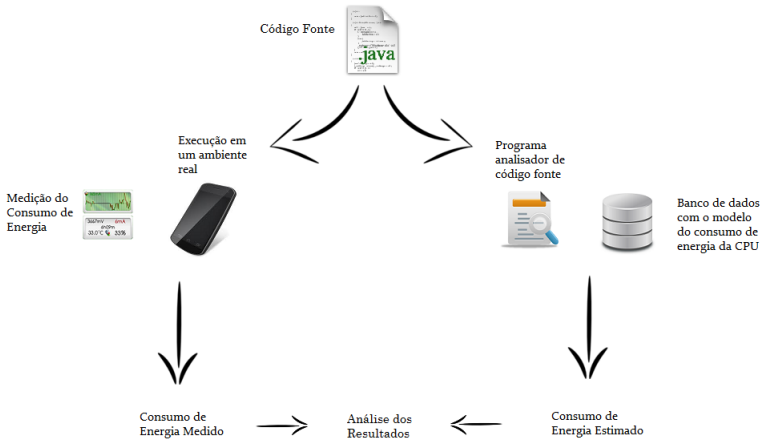

Figura 32 – Visão geral do ambiente definido para validar o modelo do dispositivo

A Figura 32 apresenta o ambiente definido para validar a metodologia de estimativa proposta. Inicialmente uma aplicação implementada na linguagem de programação Java foi

analisada por um programa contador de operações. Este programa, que utiliza o banco da dados mostrado no capítulo 4 conta as operações do programa e utiliza as equações cadastradas no banco de dados para obter a estimativa do consumo de energia da aplicação. Para comparar com o resultado real, o código fonte foi compilado e executado no Sistema Operacional do dispositivo. No caso deste projeto de mestrado, todo o ambiente foi construído utilizando-se a plataforma Java e o Sistema Android. Na execução do programa foram coletadas informações da potência média em *milliwatts* e o tempo de resposta em *millissegundos*, calculando assim o consumo de energia em *microjoules*.

Para validar as equações obtidas através da regressão, duas aplicações contendo diferentes cálculos matemáticos e operações com diversos tipos de dados foram implementadas e executadas em ambos os dispositivos apresentados anteriormente. A primeira aplicação foi construída baseada em um algoritmo de natureza altamente paralelizável da área de computação gráfica. O objetivo desse algoritmo é simular o aquecimento de uma chapa metálica em diversos pontos fazendo com que o calor se espalhe por ela. Nessa aplicação, cada pixel é representado por um elemento da matriz. Inicialmente, é dado um ponto ao qual a chapa é aquecida e o valor desse elemento é configurado para o número três enquanto os outros são todos atribuídos com o valor um. A chapa é então dividida em sub-matrizes com três linhas e três colunas e são multiplicadas entre si a cada iteração, como se o calor estivesse se propagando na chapa.

Outra aplicação utilizada para realizar a validação foi a resolução de uma integral utilizando o método trapezoidal. Esse tipo de cálculo é muito comum em aplicações científicas e dependendo dos limites inferiores e superiores na resolução da integral, podem demorar muito para processar. Da mesma forma que o algoritmo mostrado anteriormente, a resolução de uma integral pelo método trapezoidal pode ser facilmente paralelizada já que o método consiste em calcular a área abaixo do gráfico (para o caso de integral com apenas uma variável) ou o volume (para integrais com duas variáveis) em uma integral definida. Na resolução, pequenos intervalos de tamanhos iguais são definidos e têm seus respectivos volumes calculados utilizando o método trapezoidal. A soma de todos os volumes obtidos é o resultado do cálculo da integral:

$$\int_0^{10000} \int_0^{10000} (x+y^2) dydx$$

O número de operações encontradas em cada aplicação é apresentado nas Tabelas 5 e 6, onde a Tabela 5 mostra os valores para a aplicação que envolve a multiplicação de matrizes enquanto a Tabela 6 mostra o número de operações envolvidos no cálculo da integral.

Operações	Número de Operações
Soma de Doubles	5400000000
Multiplicação de Doubles	8100000000

Tabela 5 – Número de operações executadas na aplicação de simulação de dissipação de calor

Operação	Número de Operações
Soma de Inteiros	100010000
Soma de Doubles	1700000000
Subtrações de Doubles	2
Multiplicações de Doubles	1100000000
Divisões de Doubles	1100000002

Tabela 6 – Número de operações executadas pela aplicação de resolução de integral

É importante destacar que tanto na fase da execução dos *benchmarks* quanto na criação dos modelos foi contabilizado qualquer parte do código que envolveu processamento inclusive os *loops* que tinham incrementos em variáveis para controle do fluxo do programa como *for, while, do-while, switch* e *foreach*.

Os experimentos foram executados em ambos os hardwares apresentados e os resultados obtidos foram comparados aos resultados estimados utilizando as equações para os modelos de cada processador. Em todos os casos foi considerada a granularidade das operações, ou seja, quantas operações eram executadas em seguida e o tipo de dado envolvido no cálculo.

Cada aplicação foi executada quinze vezes. Esse número foi o suficiente para obter uma baixa variância e intervalo de confiança, que possibilitou a análise desses dados e a conclusão sobre os experimentos.

As Tabelas 7 e 8 mostram a média das 15 execuções considerando o consumo de energia quando da execução de ambas as aplicações descritas anteriormente para o *Dispositivo A*. Essas tabelas, mostram também o valor estimado utilizando as equações obtidas no banco de dados e finalmente o erro que a estimativa obteve em relação aos valores medidos. Em seguida, as Tabelas 9 e 10 mostram esses mesmos dados para o *Dispositivo B*. Os resultados obtidos na execução real da aplicação e na estimativa foram próximos já que a mesma mostra um aumento entre 6.43% no melhor caso e 14.24% no pior caso.

Algoritmo	Valor (µJ)	Erro da Estimativa
Média Medida	144496536.70	6.43%
Valor Estimado	153793658.00	

Tabela 7 – Resultado para ARM Cortex-A9 MPCore no *Dispositivo A*. Aplicação de Dissipação de Calor

Algoritmo	Valor (µJ)	Erro da Estimativa
Média Medida	54675170.07	9,09%
Valor Estimado	59650571.19	

Tabela 8 – Resultado para ARM Cortex-A9 MPCore no *Dispositivo A*. Aplicação do Cálculo da Integral

Outro ponto interessante é que o *Dispositivo B* obteve um maior erro na estimativa do consumo de energia em ambos os experimentos. Isso aconteceu porque esse hardware mostrou uma maior variância e obteve um maior intervalo de confiança ao executar esses experimentos.

Algoritmo	Valor (µJ)	Erro da Estimativa
Média Medida	176659481.10	14.24%
Valor Estimado	201815353.00	

Tabela 9 – Resultado para MSM7227 Cortex-A5 no *Dispositivo B*. Aplicação de Dissipação de Calor

Algoritmo	Valor (µJ)	Erro da Estimativa
Média Medida	68531656.47	9.86%
Valor Estimado	75289875.76	

Tabela 10 – Resultado para MSM7227 Cortex-A5 no *Dispositivo B* . Aplicação do Cálculo da Integral

Ainda, o *Dispositivo A* obteve um menor consumo de energia já que o mesmo trabalha em uma potência menor levando apenas um pequeno tempo a mais para executar a mesma tarefa. É possível ver o desempenho de cada aplicação na Figura 33.

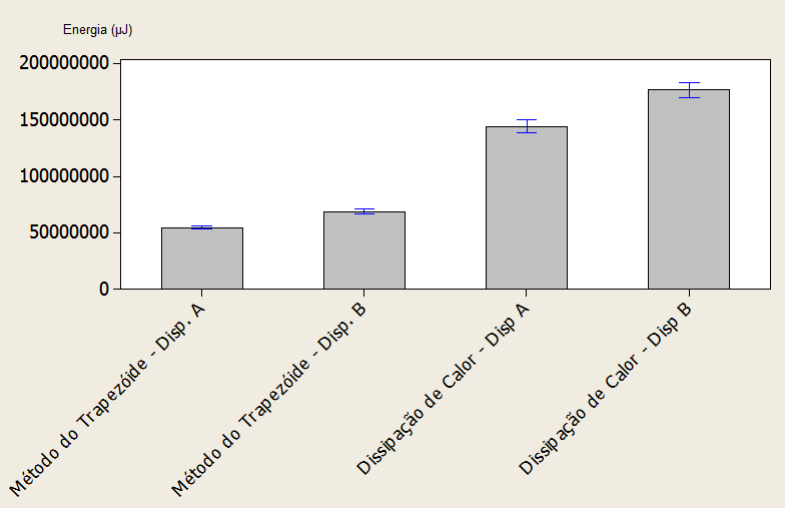

Figura 33 – Experimentos envolvendo as aplicações

5.9 Considerações Finais

Com os experimentos realizados, a metodologia proposta neste trabalho para estimativa do consumo de energia se mostrou válida, visto que no pior caso o erro da estimativa foi de 14.24%. Foi mostrado também que processadores *ARM* podem variar sua voltagem fazendo com que o consumo de energia diminua consideravelmente. No próximo capítulo são detalhadas as conclusões do projeto, contribuições e trabalhos futuros.

CAPÍTULO

6

CONCLUSÃO

6.1 Conclusões do Projeto

Nos últimos anos, com a difusão e melhoria no desempenho dos dispositivos móveis, aumentou-se a possibilidade de integração desses dispositivos em um ambiente de grade computacional e com isso introduziu-se também novos desafios nesse contexto. Este trabalho foi motivado pelo problema da limitação processamento e principalmente de bateria encontrada nesses dispositivos. Para que seja possível utilizar dispositivos móveis em uma grade computacional o escalonador deve estar ciente do trabalho necessário para a execução de uma tarefa. É inviável que um dispositivo móvel receba uma tarefa com carga de trabalho que leve muito tempo para processar ou que para ser executada com sucesso, essa tarefa necessite de uma quantidade de bateria insuficiente no momento. Dessa forma é preciso uma maneira de estimar o consumo de energia e tempo de resposta necessário para que uma determinada requisição seja atendida em um ambiente de grade computacional móvel.

É possível estimar o consumo de energia em um computador ou dispositivo móvel, entendendo o comportamento do mesmo durante a execução de um programa ou um trecho de código. Diferentes abordagens são possíveis para relacionar o consumo de energia e o código fonte. É possível olhar de uma perspectiva mais detalhista, investigando cada instrução que é executada, ou por uma abordagem mais ampla contabilizando o consumo de energia de uma chamada de uma função contendo uma razoável quantidade de código. Diferente de outras pesquisas mostradas na Seção 3.5, este trabalho utilizou uma abordagem independente de arquitetura de dispositivo e tentou relacionar as operações matemáticas mais comumente encontradas em programas científicos e seus respectivos consumos de energia.

Neste trabalho, foi mostrado também o ambiente construído que possibilitou a criação de um *profile* para cada dispositivo usado. Este ambiente foi composto por diferentes componentes. Primeiramente foram construídos programas *benchmarks* que foram executados em um

dispositivo alvo a fim de criar equações que definissem o comportamento do dispositivo sobre o consumo de energia. Os dados obtidos na execução dos *benchmarks* foram armazenados em um banco de dados relacional. Ainda, foi definido e implementado um programa contador de operações que utilizava como entrada um programa alvo, contava suas operações e calculava o consumo de energia de acordo com o modelo do dispositivo armazenado no banco de dados. O ambiente proposto pode ser facilmente estendido para que um novo dispositivo até então desconhecido passe a fazer parte de uma grade computacional. É preciso apenas executar os *benchmarks* criados para obter as equações que representam seu consumo de energia e armazenar essas informações.

6.2 Contribuições

Este trabalho contribui inicialmente com um estudo sobre grades computacionais móveis, mostra a importância de projetos voluntários e como os mesmos vem se desenvolvendo atualmente. São apresentadas também pesquisas relacionadas a medição e estimativa do consumo de energia tanto em computadores quanto em dispositivos móveis.

Nesta dissertação é proposto também um ambiente para medição e estimativa do consumo de energia suportando diversos dispositivos. Este ambiente foi construído e utilizado ao longo do projeto para coletar dados reais de dispositivos usados como casos de teste. Concluiu ainda que é possível utilizar esses dados para estimar o consumo de energia de uma aplicação mais complexa.

6.3 Trabalhos Futuros

A seguir são listados trabalhos futuros relacionados a este projeto:

- *Colocar mais dispositivos no sistema:* Para obter um ambiente mais heterogêneo como uma grade computacional real é preciso utilizar outros dispositivos e computadores portáteis de diferentes arquiteturas. Para isso é possível construir um modelo como os mostrados nessa dissertação.

- *Utilizar outras ferramentas para medição de potência e tempo de resposta:* Experimentar outras ferramentas e instrumentação sejam via software ou hardware (medição).

- *Construir um ambiente de grade:* Através de simulação ou com um ambiente real, utilizar os componentes criados neste projeto para compor um ambiente de grade computacional e pesquisar diferentes algoritmos de escalonamento que utilizariam as métricas apresentadas em consideração.

- *Utilizar outras formas de regressão de dados:* A fim de diminuir o erro na estimativa do consumo de energia é importante que sejam estudadas diferentes formas de regressão para obter as equações de cada operação.

- *Considerar também códigos de chamadas de sistema:* Em um ambiente real os nós se comunicam tanto com mensagens de controle quanto para enviar e receber os dados de resultados das tarefas. Dessa forma, é preciso criar *benchmarks* com chamadas do sistema para que se possa considerar o consumo de energia das mesmas em uma aplicação paralela.

REFERÊNCIAS

AGUILERA, J.; GALLEGO, F.; SILVA, C. System on chip (soc): New generation of video surveillance systems. In: **Security Technology (ICCST), 2014 International Carnahan Conference on**. [S.l.: s.n.], 2014. p. 1–5. Citado na página 18.

AMERICA, I. O. Jsr-000924 java® virtual machine specification. In: . [S.l.: s.n.], 2007. Citado 2 vezes nas páginas 17 e 31.

ANDERSON, D. P. Boinc: A system for public-resource computing and storage. In: **Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing**. Washington, DC, USA: IEEE Computer Society, 2004. (GRID '04), p. 4–10. ISBN 0-7695-2256-4. Disponível em: <http://dx.doi.org/10.1109/GRID.2004.14>. Citado na página 14.

ANDERSON, D. P.; COBB, J.; KORPELA, E.; LEBOFSKY, M.; WERTHIMER, D. Seti@home: An experiment in public-resource computing. **Commun. ACM**, ACM, New York, NY, USA, v. 45, n. 11, p. 56–61, nov. 2002. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/581571.581573>. Citado na página 14.

ARM. **ARM Architeture**. 2015. <http://www.arm.com/products/processors/instruction-set-architectures/index.php>. Acessado: 02/02/2015. Citado na página 18.

ASHMORE, S.; MAKKI, S. K. Imissar: an intelligent, mobile middleware solution for secure automatic reconfiguration of applications, utilizing a feature model approach. In: **Proceedings of the 5th International Conference on Ubiquitous Information Management and Communication**. New York, NY, USA: ACM, 2011. (ICUIMC '11), p. 58:1–58:7. ISBN 978-1-4503-0571-6. Disponível em: <http://doi.acm.org/10.1145/1968613.1968684>. Citado na página 13.

BADAM, S.; FISHER, E.; ELMQVIST, N. Munin: A peer-to-peer middleware for ubiquitous analytics and visualization spaces. **Visualization and Computer Graphics, IEEE Transactions on**, v. 21, n. 2, p. 215–228, Feb 2015. ISSN 1077-2626. Citado na página 13.

BAZZAZ, M.; SALEHI, M.; EJLALI, A. An accurate instruction-level energy estimation model and tool for embedded systems. **Instrumentation and Measurement, IEEE Transactions on**, v. 62, n. 7, p. 1927–1934, 2013. ISSN 0018-9456. Citado na página 19.

BEBERG, A.; ENSIGN, D.; JAYACHANDRAN, G.; KHALIQ, S.; PANDE, V. Folding@home: Lessons from eight years of volunteer distributed computing. In: **Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on**. [S.l.: s.n.], 2009. p. 1–8. ISSN 1530-2075. Citado na página 15.

BERRY DJAOUI, G. **GFD-I.080.The Open Grid Services Architecture**. [S.l.], 2006. Edited by I. Foster, H. Kishimoto, A. Savva. Citado na página 6.

BOINCSTATS. **Project Stats info**. 2016. <http://boincstats.com/en/stats/projectStatsInfo>. Acessado: 08/01/2016. Citado na página 14.

BORRO, L. C.; ROCHA FILHO, G. P.; BRUSCHI, S. M. Proposição e análise de um algoritmo ciente de consumo de energia para grades móveis. In: **XLVI Simpósio Brasileiro de Pesquisa Operacional (SBPO)**. [S.l.: s.n.], 2014. p. 1–12. Citado 2 vezes nas páginas 2 e 3.

BRUNEO, D.; PULIAFITO, A.; SCARPA, M. Mobile Middleware: Definition and Motivations. **The Handbook of Mobile Middleware**, p. 145–167, 2007. Disponível em: <http://www.cs.cmu.edu/~jhm/Readings/MobileMiddleware.pdf>. Citado 2 vezes nas páginas 8 e 9.

BUSCHING, F.; SCHILDT, S.; WOLF, L. Droidcluster: Towards smartphone cluster computing – the streets are paved with potential computer clusters. In: **Distributed Computing Systems Workshops (ICDCSW), 2012 32nd International Conference on**. [S.l.: s.n.], 2012. p. 114–117. ISSN 1545-0678. Citado na página 1.

CHOI, Y.-S.; JEON, Y.-J.; PARK, S.-H. A study on sensor nodes attestation protocol in a wireless sensor network. In: **Advanced Communication Technology (ICACT), 2010 The 12th International Conference on**. [S.l.: s.n.], 2010. v. 1, p. 574–579. ISSN 1738-9445. Citado na página 7.

CIRANI, S.; FERRARI, G.; IOTTI, N.; PICONE, M. The iot hub: a fog node for seamless management of heterogeneous connected smart objects. In: **Sensing, Communication, and Networking - Workshops (SECON Workshops), 2015 12th Annual IEEE International Conference on**. [S.l.: s.n.], 2015. p. 1–6. Citado na página 2.

CORRAL, L.; GEORGIEV, A.; SILLITTI, A.; SUCCI, G. A method for characterizing energy consumption in android smartphones. In: **Green and Sustainable Software (GREENS), 2013 2nd International Workshop on**. [S.l.: s.n.], 2013. p. 38–45. Citado na página 23.

COULOURIS, G. F. **Distributed Systems: Concepts and Design, 5/e**. 5. ed. Boston, Massachusetts: Addison-Wesley, 2012. 1067 p. ISBN 9780132143011. Disponível em: <http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:DISTRIBUTED+SYSTEMS+Concepts+and+Design.> Citado na página 8.

DARGIE, W. A stochastic model for estimating the power consumption of a processor. **Computers, IEEE Transactions on**, v. 64, n. 5, p. 1311–1322, May 2015. ISSN 0018-9340. Citado na página 36.

DATTA, S.; BONNET, C.; HAERRI, J. Fog computing architecture to enable consumer centric internet of things services. In: **Consumer Electronics (ISCE), 2015 IEEE International Symposium on**. [S.l.: s.n.], 2015. p. 1–2. Citado na página 2.

FEI, Y.; RAVI, S.; RAGHUNATHAN, A.; JHA, N. Energy-optimizing source code transformations for os-driven embedded software. In: **VLSI Design, 2004. Proceedings. 17th International Conference on**. [S.l.: s.n.], 2004. p. 261–266. Citado na página 19.

FOSTER, I. The physiology of the grid: An open grid services architecture for distributed systems integration. In: . [S.l.: s.n.], 2002. Citado 2 vezes nas páginas 6 e 7.

____. Globus toolkit version 4: Software for service-oriented systems. In: **Proceedings of the 2005 IFIP International Conference on Network and Parallel Computing**. Berlin, Heidelberg: Springer-Verlag, 2005. (NPC'05), p. 2–13. Disponível em: <http://dx.doi.org/10.1007/11577188_2>. Citado na página 7.

FOSTER, I.; KESSELMAN, C.; TUECKE, S. The anatomy of the grid: Enabling scalable virtual organizations. **Int. J. High Perform. Comput. Appl.**, Sage Publications, Inc., Thousand Oaks, CA, USA, v. 15, n. 3, p. 200–222, 2001. ISSN 1094-3420. Disponível em: <http://dx.doi.org/10.1177/109434200101500302>. Citado na página 7.

FOUNDATION, I. F. S. **GCC Documentation**. Dissertação (Mestrado), 2013. Disponível em: <http://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html>. Citado na página 19.

FRANKE, D.; KOWALEWSKI, S.; WEISE, C. A Mobile Software Quality Model. In: **Quality Software (QSIC), 2012 12th International Conference on**. [S.l.: s.n.], 2012. p. 154–157. ISSN 1550-6002. Citado na página 9.

GARTNER. **Gartner Says Tablet Sales Continue to Be Slow in 2015**. 2015. <http://www.gartner.com/newsroom/id/2954317>. Acessado: 02/11/2015. Citado na página 1.

GORTON, I. **Essential Software Architecture**. [S.l.]: Springer, 2006. Citado na página 9.

HAO, S.; LI, D.; HALFOND, W.; GOVINDAN, R. Estimating android applications' cpu energy usage via bytecode profiling. In: **Green and Sustainable Software (GREENS), 2012 First International Workshop on**. [S.l.: s.n.], 2012. p. 1–7. Citado 2 vezes nas páginas 20 e 36.

IEEE Standard Glossary of Software Engineering Terminology. [S.l.], 1990. Citado na página 10.

ISAIADIS, S.; GETOV, V. Integrating mobile devices into the grid: Design considerations and evaluation. In: CUNHA, J.; MEDEIROS, P. (Ed.). **Euro-Par 2005 Parallel Processing**. Springer Berlin Heidelberg, 2005, (Lecture Notes in Computer Science, v. 3648). p. 1080–1088. ISBN 978-3-540-28700-1. Disponível em: <http://dx.doi.org/10.1007/11549468_118>. Citado na página 7.

JINDAL, A.; PATHAK, A.; HU, Y. C.; MIDKIFF, S. Hypnos: Understanding and treating sleep conflicts in smartphones. In: **Proceedings of the 8th ACM European Conference on Computer Systems**. New York, NY, USA: ACM, 2013. (EuroSys '13), p. 253–266. ISBN 978-1-4503-1994-2. Disponível em: <http://doi.acm.org/10.1145/2465351.2465377>. Citado na página 24.

JOHNSEN, F.; BLOEBAUM, T.; EGGUM, D. Efficient soap messaging for android. In: **Military Communications and Information Systems (ICMCIS), 2015 International Conference on**. [S.l.: s.n.], 2015. p. 1–9. Citado na página 20.

KAMAL, R.; RAZZAQUE, M.; HONG, C. seon. Grid middleware for invivo sensor nodes. In: **Information Networking (ICOIN), 2011 International Conference on**. [S.l.: s.n.], 2011. p. 200–205. ISSN 1976-7684. Citado na página 12.

KAPETANAKIS, K.; PANAGIOTAKIS, S. Efficient energy consumption's measurement on android devices. In: **Informatics (PCI), 2012 16th Panhellenic Conference on**. [S.l.: s.n.], 2012. p. 351–356. Citado na página 20.

KEHTARNAVAZ, N.; PARRIS, S.; SEHGAL, A. Using smartphones as mobile implementation platforms for applied digital signal processing courses. In: **Signal Processing and Signal Processing Education Workshop (SP/SPE), 2015 IEEE**. [S.l.: s.n.], 2015. p. 313–318. Citado na página 1.

KHAN, J.; BILAVARN, S.; BELLEUDY, C. Energy analysis of a dvfs based power strategy on arm platforms. In: **Faible Tension Faible Consommation (FTFC), 2012 IEEE**. [S.l.: s.n.], 2012. p. 1–4. Citado na página 39.

KISHIMOTO FUJITSU, T. **Defining the Grid: A Roadmap for OGSA Standards**. [S.l.], 2005. Citado na página 7.

KRAUTER, K.; BUYYA, R.; MAHESWARAN, M. A taxonomy and survey of grid resource management systems for distributed computing. **Softw. Pract. Exper.**, John Wiley & Sons, Inc., New York, NY, USA, v. 32, n. 2, p. 135–164, 2002. ISSN 0038-0644. Disponível em: <http://dx.doi.org/10.1002/spe.432>. Citado na página 6.

LI, D.; HAO, S.; HALFOND, W. G. J.; GOVINDAN, R. Calculating source line level energy information for android applications. In: **Proceedings of the 2013 International Symposium on Software Testing and Analysis**. New York, NY, USA: ACM, 2013. (ISSTA 2013), p. 78–89. ISBN 978-1-4503-2159-4. Disponível em: <http://doi.acm.org/10.1145/2483760.2483780>. Citado na página 20.

LIN, Y.; SHEN, H. Cloud fog: Towards high quality of experience in cloud gaming. In: **Parallel Processing (ICPP), 2015 44th International Conference on**. [S.l.: s.n.], 2015. p. 500–509. ISSN 0190-3918. Citado na página 2.

LIU, X.; TAYLOR, S. J. E.; MUSTAFEE, N.; WANG, J.; GAO, Q.; GILBERT, D. Speeding up systems biology simulations of biochemical pathways using condor. **Concurr. Comput. : Pract. Exper.**, John Wiley and Sons Ltd., Chichester, UK, v. 26, n. 17, p. 2727–2742, dez. 2014. ISSN 1532-0626. Disponível em: <http://dx.doi.org/10.1002/cpe.3161>. Citado na página 15.

LUO, G.; GUO, B.; SHEN, Y.; LIAO, H.; REN, L. Analysis and optimization of embedded software energy consumption on the source code and algorithm level. In: **Embedded and Multimedia Computing, 2009. EM-Com 2009. 4th International Conference on**. [S.l.: s.n.], 2009. p. 1–5. Citado na página 19.

MA, J.; YU, H.; GONG, X.; ZHANG, X. Research on online measurement method of smartphone energy consumption. In: **P2P, Parallel, Grid, Cloud and Internet Computing (3PGCIC), 2014 Ninth International Conference on**. [S.l.: s.n.], 2014. p. 443–447. Citado na página 21.

MAIRIZA, D.; ZOWGHI, D.; NURMULIANI, N. An investigation into the notion of non-functional requirements. In: **Proceedings of the 2010 ACM Symposium on Applied Computing**. New York, NY, USA: ACM, 2010. (SAC '10), p. 311–317. ISBN 978-1-60558-639-7. Disponível em: <http://doi.acm.org/10.1145/1774088.1774153>. Citado 2 vezes nas páginas 9 e 10.

MAKKI, S. K.; SRIRANGAM, N. B.; AISWARYA, V. S.; YU, S. Utilizing intelligent middleware for reconfiguration of applications on android. In: **Proceedings of the 5th international conference on Convergence and hybrid information technology**. Berlin, Heidelberg: Springer-Verlag, 2011. (ICHIT'11), p. 81–89. ISBN 978-3-642-24081-2. Disponível em: <http://dl.acm.org/citation.cfm?id=2045005.2045016>. Citado na página 13.

MASINDE, M.; BAGULA, A.; NDEGWA, V. Mobigrid: A middleware for integrating mobile phone and grid computing. In: **Network and Service Management (CNSM), 2010 International Conference on**. [S.l.: s.n.], 2010. p. 523–526. Citado na página 10.

NGUESSAN, D.; SIDNEI, J.; MARTINI, C.; PAULO, S. a. Study and Implementation of a Solution to Security Management for Mobile Environments Based on Tuple. p. 2014–2020, 2008. Citado na página 13.

PAGEL, M.; CARLSON, D. Ambient control: A mobile framework for dynamically remixing the internet of things. In: **World of Wireless, Mobile and Multimedia Networks (WoWMoM), 2015 IEEE 16th International Symposium on a**. [S.l.: s.n.], 2015. p. 1–9. Citado na página 13.

PATHAK, A.; HU, Y. C.; ZHANG, M. Where is the energy spent inside my app?: fine grained energy accounting on smartphones with eprof. In: **Proceedings of the 7th ACM european conference on Computer Systems**. New York, NY, USA: ACM, 2012. (EuroSys '12), p. 29–42. ISBN 978-1-4503-1223-3. Disponível em: <http://doi.acm.org/10.1145/2168836.2168841>. Citado na página 24.

PERERA, C.; JAYARAMAN, P.; ZASLAVSKY, A.; GEORGAKOPOULOS, D.; CHRISTEN, P. Mosden: An internet of things middleware for resource constrained mobile devices. In: **System Sciences (HICSS), 2014 47th Hawaii International Conference on**. [S.l.: s.n.], 2014. p. 1053–1062. Citado na página 13.

PFISTER, G. F. **In Search of Clusters (2Nd Ed.)**. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1998. ISBN 0-13-899709-8. Citado na página 5.

PHAN, T.; HUANG, L.; DULAN, C. Challenge: Integrating mobile wireless devices into the computational grid. In: **Proceedings of the 8th Annual International Conference on Mobile Computing and Networking**. New York, NY, USA: ACM, 2002. (MobiCom '02), p. 271–278. ISBN 1-58113-486-X. Disponível em: <http://doi.acm.org/10.1145/570645.570679>. Citado na página 1.

POPESCU, A.; ERMAN, D.; VOGELEER, K. de; POPESCU, A.; FIEDLER, M. Network performance engineering. In: KOUVATSOS, D. D. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2011. cap. ROMA: a middleware framework for seamless handover, p. 784–794. ISBN 978-3-642-02741-3. Disponível em: <http://dl.acm.org/citation.cfm?id=1996629.1996669>. Citado na página 9.

PROCHKOVA, I.; SINGH, V.; NURMINEN, J. Energy cost of advertisements in mobile games on the android platform. In: **Next Generation Mobile Applications, Services and Technologies (NGMAST), 2012 6th International Conference on**. [S.l.: s.n.], 2012. p. 147–152. ISSN 2161-2889. Citado na página 20.

RODRIGUEZ, J.; MATEOS, C.; ZUNINO, A. Are smartphones really useful for scientific computing? In: CIPOLLA-FICARRA, F.; VELTMAN, K.; VERBER, D.; CIPOLLA-FICARRA, M.; KAMMüLLER, F. (Ed.). **Advances in New Technologies, Interactive Interfaces and Communicability**. Springer Berlin Heidelberg, 2012, (Lecture Notes in Computer Science, v. 7547). p. 38–47. ISBN 978-3-642-34009-3. Disponível em: <http://dx.doi.org/10.1007/978-3-642-34010-9_4>. Citado na página 2.

RODRIGUEZ, J. M.; ZUNINO, A.; CAMPO, M. Introducing mobile devices into grid systems; a survey. **Int. J. Web Grid Serv.**, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 7, n. 1, p. 1–40, feb 2011. ISSN 1741-1106. Disponível em: <http://dx.doi.org/10.1504/IJWGS.2011.038386>. Citado na página 8.

RUMI, M.; ASADUZZAMAN; HASAN, D. H. Cpu power consumption reduction in android smartphone. In: **Green Energy and Technology (ICGET), 2015 3rd International Conference on**. [S.l.: s.n.], 2015. p. 1–6. Citado na página 24.

SARKAR, S.; CHATTERJEE, S.; MISRA, S. Assessment of the suitability of fog computing in the context of internet of things. **Cloud Computing, IEEE Transactions on**, PP, n. 99, p. 1–1, 2015. ISSN 2168-7161. Citado na página 2.

SHARKEY, J. **Coding for Life - Battery Life, That Is**. [S.l.], 2009. Disponível em: <https://dl.google.com/io/2009/pres/W_0300_CodingforLife-BatteryLifeThatIs.pdf>. Citado na página 20.

SHEN, X. Mobile crowdsourcing [editor's note]. **Network, IEEE**, v. 29, n. 3, p. 2–3, May 2015. ISSN 0890-8044. Citado na página 2.

SHIH, C.-S.; WANG, Y.-H.; CHANG, N. Multi-tier elastic computation framework for mobile cloud computing. In: **Mobile Cloud Computing, Services, and Engineering (MobileCloud), 2015 3rd IEEE International Conference on**. [S.l.: s.n.], 2015. p. 223–232. Citado na página 13.

SIMUNIC, T.; BENINI, L.; MICHELI, G. D.; HANS, M. Source code optimization and profiling of energy consumption in embedded systems. In: **System Synthesis, 2000. Proceedings. The 13th International Symposium on**. [S.l.: s.n.], 2000. p. 193–198. ISSN 1080-1820. Citado na página 19.

SINGH, M. P.; JAIN, M. K. Article: Evolution of processor architecture in mobile phones. **International Journal of Computer Applications**, v. 90, n. 4, p. 34–39, March 2014. Full text available. Citado na página 18.

TANENBAUM, M. V. S. A. S. **Sistemas Distribuídos: Princípios e Paradigmas**. Second. [S.l.]: Pearson Prentice Hall, 2007. Citado 2 vezes nas páginas 6 e 26.

TEIXEIRA, F. **Grid Anywhere: Um middleware extensível para grades computacionais desktop**. Tese (Doutorado) — LasDPC, Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2012. Citado na página 11.

VARAPRASAD, G.; RAJU, G. A middleware for distributed system in heterogeneous wireless networks. **... and Distributed ...**, 2005. Disponível em: <http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1524395>. Citado na página 12.

VERBELEN, T.; SIMOENS, P.; TURCK, F. D.; DHOEDT, B. AIOLOS: Middleware for improving mobile application performance through cyber foraging. **Journal of Systems and Software**, v. 85, n. 11, p. 2629–2639, 2012. ISSN 01641212. Disponível em: <http://dx.doi.org/10.1016/j.jss.2012.06.011>. Citado na página 13.

VIEIRA, A.; DEBASTIANI, D.; AGOSTINI, L.; MARQUES, F.; MATTOS, J. Performance and energy consumption analysis of embedded applications based on android platform. In: **Computing System Engineering (SBESC), 2012 Brazilian Symposium on**. [S.l.: s.n.], 2012. p. 59–64. ISSN 2324-7886. Citado na página 20.

WILKE, C.; GOTZ, S.; RICHLY, S. Jouleunit: A generic framework for software energy profiling and testing. In: **Proceedings of the 2013 Workshop on Green in/by Software Engineering**. New York, NY, USA: ACM, 2013. (GIBSE '13), p. 9–14. ISBN 978-1-4503-1866-2. Disponível em: <http://doi.acm.org/10.1145/2451605.2451610>. Citado na página 21.

ZHANG, L.; TIWANA, B.; QIAN, Z.; WANG, Z.; DICK, R. P.; MAO, Z. M.; YANG, L. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In: **Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis**. New York, NY, USA: ACM, 2010. (CODES/ISSS '10), p. 105–114. ISBN 978-1-60558-905-3. Disponível em: <http://doi.acm.org/10.1145/1878961.1878982>. Citado 2 vezes nas páginas 21 e 39.

ZHOU, X.; GUO, B.; SHEN, Y.; LI, Q. Design and implementation of an improved c source-code level program energy model. In: **Embedded Software and Systems, 2009. ICESS '09. International Conference on**. [S.l.: s.n.], 2009. p. 490–495. Citado na página 19.

GLOSSÁRIO

ARM: É uma arquitetura de processador de 32 bits usada principalmente em sistemas embarcados. São processadores que têm o objetivo de atingir a máxima eficiência por ciclo de forma a minimizar o consumo de energia.

CPU: *Central processing unit* ou Unidade central de processamento. É a parte de um sistema computacional que executa as instruções do código de um programa.

DVFS: O Dimensionamento Dinâmico de Tensão é uma técnica em computadores no qual a voltagem pode ser aumentada ou diminuída de acordo com as necessidades momentâneas fazendo com que seja possível economizar energia elétrica em alguns casos.

E/S: Em computação, entrada e saída, ou E/S é a comunicação entre o sistema de processamento ou seja a CPU e algum periférico como por exemplo, um mouse, teclado ou placa de rede.

GDB: O GNU Debugger, mais conhecido por GDB, é um depurador do sistema operacional GNU/Linux. É utilizado na depuração de diversas linguagens como C, C++ e Java.

GPS: O Sistema de Posicionamento Global (GPS) é um sistema criado para navegação e utiliza satélites para definir o posicionamento dos usuários e dos locais.

JIT: Do inglês *Just in Time* é um termo em ciência da computação, utilizado para designar um compilador que faz a tradução de um código para a linguagem nativa do sistema ao executar o programa.

JSON: JSON é um acrônimo para *JavaScript Object Notation*. Consiste em um formato para troca de informações onde os dados são representados em texto.

JVM: Do inglês *Java Virtual Machine* é uma Máquina Virtual, ou seja, um programa que carrega e executa os aplicativos Java, convertendo os bytecodes em código executável de máquina em tempo de execução.

LCD: Um display de cristal liquido ou LCDs são displays utilizados para mostrar imagens ou informações para um usuário. Podem ser encontrados em monitores ou telas de *smartphones* atuais.

MPEG-1 Layer III: É um codec de áudio que utiliza compressão e de-compressão com perdas.

OGSA: Do inglês *Open Grid Services Architecture* descreve uma arquitetura orientada a serviços para uso científico ou empresarial.

RISC: *Reduced Instruction Set Computer* ou *Computador com um Conjunto Reduzido de Instruções*, é uma arquitetura de processadores que fornece poucas tipos de instruções através do hardware e instruções mais complexas devem ser implementadas utilizando um conjunto das suas instruções.

SDK: Um *Software Development Kit* ou Kit de Desenvolvimento de Software é composto de ferramentas que proporcionam a um programador desenvolver um software para alguma plataforma ou hardware específico.

SOAP: Em português *Protocolo Simples de Acesso a Objetos* ou em inglês *Simple Object Access Protocol* é um protocolo utilizado principalmente em serviços web para troca de informações. Ele se baseia em marcações XML para o formato de mensagem.

UDP: O *User Datagram Protocol* (UDP) é um protocolo simples da camada de transporte. Diferente de outros protocolos da camada de transporte esse protocolo não oferece controle de fluxo e confiabilidade no transporte de dados.

Wifi: Wi-Fi é uma rede do tipo local em que os computadores podem se conectar e trocara informações. É muito utilizada por dispositivos móveis também e utiliza frequências em torno de 2.4 Ghz para transmitir os dados.

WSDL: *Web Services Description Language* é uma linguagem que utiliza o próprio XML para descrever os parâmetros de entrada e de retorno de um serviço web.