Performance and Energy Consumption Analysis of Embedded Applications based on Android Platform

Andrws Vieira, Daniel Debastiani, Luciano Agostini, Felipe Marques, Julio C. B. Mattos
Grupo de Arquitetura e Circuitos Integrados – GACI
Universidade Federal de Pelotas – UFPel
Pelotas, Brasil
aavieira, dsdebastiani, agostini, felipem, julius {@inf.ufpel.edu.br}

***Abstract* — This paper presents an analysis of embedded applications based on Android Platform. Analyzing performance and energy consumption from different algorithmic versions this work tries to find a performance and energy pattern for the paradigm used in each used algorithm. Thus, the developer can select the best algorithm version for each application based on the requirements. Android is a Linux based mobile operating system developed by Google in conjunction with the Open Handset Alliance. Nowadays, Android is the world's leading smartphone platform. The Android Platform provides a rich development environment (SDK) including an API, development and debug tool and so on. However, during the development process it is necessary to handle with the system functionalities and it must at the same time help the handling of the embedded systems tight constraints, like performance, energy and power. This paper shows some performance and energy consumption results for a set of applications using different algorithmic versions.**

Keywords —Android, Performace, Energy, Analysis.

I.INTRODUÇÃO

O Android é uma plataforma de desenvolvimento para aplicativos móveis baseada em um sistema operacional Linux[1], sendo um projeto de código aberto e liderado pela Google. Por ter seu código fonte aberto e pelo tipo de licença a plataforma se torna flexível, permitindo que customizações sejam feitas sem que os fabricantes precisem compartilhar tais alterações.

O desenvolvimento de aplicações para Android é facilitado devido ao kit de desenvolvimento que disponibiliza ferramentas e APIs necessárias para o desenvolvimento de aplicações e fornecem fácil integração com diversos recursos físicos disponíveis no aparelho.

Durante o desenvolvimento de aplicações móveis se faz necessário à análise de requisitos não funcionais como desempenho, consumo de potência e energia, tendo em vista que estas aplicações são executadas em aparelhos com bateria.

O foco deste trabalho está na avaliação de aplicações para Android, quanto ao desempenho, dissipação de potência e consumo de energia, em diferentes versões algorítmicas. Assim diagnosticando a melhor versão de um determinado algoritmo para um fim especifico.

II. ESTADO DA ARTE

Prolongar a vida útil da bateria em dispositivos móveis sempre foi um desafio. Muitos pesquisadores têm como objeto de estudo metodologias de otimização e eficiência em energia[2]. A complexidade dos dispositivos modernos está crescendo rapidamente, aumentando sua capacidade computacional. Esse aumento na capacidade de processamento tende a aumentar o consumo de energia. Entretanto, esse do consumo é indesejado pois queremos estender o tempo de vida útil da bateria do dispositivo[3].

A previsão é que até 2013 dispositivos móveis como smartphones e tablets vão ultrapassar os PCs como os dispositivos mais utilizados para acesso à internet [4]. Os smartphones mais modernos são capazes de rodar vídeo e áudio em alta definição, prover acesso à internet de alta velocidade, permitem fotografar e gravar vídeos de alta qualidade – HD – além de possuírem interface com GPS e sensores como acelerômetro.

Esta área de pesquisa que visa um balanço entre energia consumida e desempenho em sistemas embarcados está em alta devido ao crescente numero de smartphones e tablets, e pelo fato de que o Android vem se destacando cada vez mais entre os sistemas embarcados nestes dispositivos. Existem vários trabalhos de pesquisa que visam apresentar sistemas que realizam estimativas em tempo real do consumo de energia em dispositivos móveis Android. PowerRunner [5] e PowerTutor [6] são resultados de pesquisas com este foco.

PowerRunner é um software de gerenciamento automatizado de energia que prevê e otimiza o consumo de energia em dispositivos móveis, utilizando aprendizagem, kernels Linux e uma API de baixo nível para o gerenciamento de energia em tempo de execução. Kernels Linux modernos definem uma estrutura para o código de gerenciamento de energia que facilitam desligar e ligar individualmente um componente do sistema em tempo de execução. Através do monitoramento cuidadoso e aprendizagem do comportamento dos componentes do sistema usados por várias atividades do usuário, PowerRunner analisa e estima os recursos necessários por uma Acitivity no Android. A fim de maximizar a experiência do usuário, minimizar os requisitos de energia e maximizar a economia de energia aplicando combinações de várias otimizações em tempo de execução. O projeto provou economizar até 25% de energia de forma consistente [5].

978-0-7695-4929-3/12 $26.00 © 2012 IEEE
DOI 10.1109/SBESC.2012.20

PowerTutor é um sistema de estimativa de consumo de energia em tempo real implementado para smartphones com plataforma Android. PowerTutor fornece estimativas precisas de consumo de energia em tempo real para os componentes de hardware como CPU, display LCD, GPS, Wi-Fi, áudio e interfaces de rede de celular.

Para analisar o desempenho de uma aplicação no Android existe uma ferramenta chamada Traceview. Traceview é um visualizador gráfico para logs de execução incluso na classe Debug do Android, com finalidade de registrar informações de rastreamento em uma aplicação. Além disso o Traceview também ajuda a depurar uma aplicação e analisar seu desempenho [7]. A partir desta ferramenta é possível encontrar os gargalos de desempenho de um aplicativo e assim ter margem para maximizar o desempenho de tal aplicação.

O foco principal deste trabalho está em utilizar ferramentas para determinar o impacto de alterações de projeto de software no consumo de energia e potência dissipada e desempenho.

Na próxima seção será apresentado um breve estado da arte. Na terceira seção serão descritos alguns algoritmos utilizados em nossos experimentos. Em seguida na quarta seção serão apresentados os resultados obtidos. Na quinta seção serão apresentadas as conclusões e por fim na sexta seção as referências.

III. ESTUDOS DE CASOS

A etapa de análise foi dividia em três momentos sendo estes: análise de algoritmos recursivos e iterativos de mesma complexidade, análise de algoritmos que fazem mais uso de memoria versus algoritmos que fazem mais uso de CPU e análise de alguns algoritmos de ordenação populares.

O processo de escolhe dos algoritmos analisados se deu da seguinte maneira:

A. Algoritmos recursivos versus interativos de mesma complexidade

Para o desenvolvimento desta parte do trabalho, foi necessário pesquisar algoritmos que possuíssem a mesma complexidade nas versões iterativa e recursiva, para então fazer a análise de desempenho e consumo de energia sobre cada par de algoritmos. O primeiro algoritmo implementado foi o Insertion Sort, que possui complexidade assintótica $O(n^2)$, como pode ser comprovado a partir da equação de recorrência(1).

$$\begin{cases} T(1) = C \\ T(n) = T(n-1) + n \end{cases} \quad (1)$$

A partir da equação de recorrência, pode-se aplicar o método de substituição para provar que a complexidade do Insertion Sort recursivo é $O(n^2)$. No método de substituição supõe-se um limite hipotético para então usar a indução matemática para provar que a suposição está correta [8].

Na implementação iterativa pode-se notar a partir da figura 1 que a complexidade superior assintótica é limitada por $O(n^2)$, pois no pior caso os dois laços serão executados *n* vezes. O outro algoritmo selecionado para esta etapa de analise foi o Fatorial que, usando os mesmos métodos, pôde ser comprovado que sua complexidade é O (n), tanto para versão iterativa quando para versão recursiva.

```
public int insertionSort (int[] array, int n){
    int i, j, chave;
    for(j=1; j<n; j++){
        chave = array[j];
        i = j-1;
        while(i >= 0 && array[i] > chave){
            array[i+1] = array[i];
            i--;
            }
        array[i+1] = chave;
    }
    return 1;
}
```

Figura 1. Insertion Sort Iterativo.

Casos de testes foram estabelecidos para analisar o desempenho e o consumo de energia destes algoritmos na plataforma Android. Para gerar uma entrada de grande tamanho para ser ordenada pelo algoritmo Insertion Sort, foi desenvolvido um script utilizando a linguagem Lua [9]. Este script gerou números aleatórios inteiros positivos no intervalo de zero a duzentos. A mesma entrada foi usada para executar as versões recursiva e iterativa do algoritmo. Na versão recursiva foi detectado um problema, quando o número de chamadas recursivas beirava a trezentos, gerava um estouro de pilha no Android, dificultando assim a análise e a comparação dos resultados. Para solucionar este problema a função que executava os algoritmos foi chamada várias vezes para aumentar o tempo de execução da aplicação.

No algoritmo Fatorial recursivo o mesmo problema de estouro de pilha ocorreu. Neste caso o problema foi solucionado usando um conjunto de entradas com "N" menor igual a trezentos. Tal conjunto serviu de entrada tanto para a versão recursiva quanto iterativa.

B. Algoritmos que fazem mais uso de memoria versus algoritmos que fazem mais uso de CPU

Para esta análise o algoritmo da função Seno se enquadrou perfeitamente exibindo um comportamento desejado para esta análise, pois é possível armazenar determinados resultados do seno em uma tabela e posteriormente acessa-los por uma função Hash. Foram armazenados os valores do Seno de zero a noventa em um intervalo de um decimo, assim totalizando uma tabela com novecentos valores.

Já para calcular o seno(x) consideramos a série infinita do seno conforme a equação (2), porém apenas considerando os primeiros termos.

$$Seno(x) = \frac{x}{1!} - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots \quad (2)$$

O algoritmo que utilizamos, como pode ser observado na fig. 2, representa fielmente a série (2) e gera como resultado o seno de zero a noventa em intervalos de um decimo, igualmente ao seno tabelado.

```
angulo = (double) 0;
while (angulo <= 90) {
    rad = (double) (angulo * Math.PI / 180);
    soma = 0;
    sinal = 1;
    expoente = 1;
    fat = 1;
    b = 1;
    repeticao = 10;
    for (a = 1; a <= repeticao; a += 1) {
        for (c = 1; c <= b; c += 1) {
            fat = fat * c;
        }
        soma = (double) (soma + sinal
                * Math.pow(rad, expoente) / fat);
        expoente += 2;
        b += 2;
        fat = 1;
        sinal = -sinal;
    }
    angulo = (double) (angulo + 0.1);
}
```

Figura 2. Algoritmo do Seno Calculado.

C. Algoritmos de ordenação

Um problema computacional que surge com frequência na prática é o problema da ordenação, que consiste basicamente em ordenar uma sequencia de números em ordem crescente. Formalmente o problema da ordenação pode ser definido da seguinte forma:

Entrada: uma sequencia de n números ⌊a1, a2, ...,an⌋.

Saída: Uma permutação (reordenação) ⌊a1', a2',...,an'⌋ da sequência de entrada tal que a1' ⩽ a2' ⩽ ... ⩽ an.

A operação de ordenação é fundamental para a Ciência da Computação, onde muitos programas a utilizam como uma etapa intermediária de sua execução e, como resultado, um grande número de algoritmos de ordenação tem sido desenvolvido.

Um algoritmo de ordenação pode ser classificado pela complexidade computacional da comparação de elementos em termos do tamanho da sequencia *n*, pela complexidade computacional de trocas (permutações) realizadas, uso de memória e de outros recursos computacionais, uso de recursividade, estabilidade do algoritmo, método geral (inserção, troca, seleção) e sua adaptabilidade ao padrão da sequencia de entrada.

Em geral um algoritmo é considerado mais eficiente que outro se o tempo de execução em seu pior caso apresenta uma ordem de crescimento mais baixa. A ordem de crescimento está diretamente relacionada ao tamanho da sequência de entrada *n*, e permite a comparação relativa do desempenho entre algoritmos. Quando observado tamanhos de entradas suficientemente grandes para tornar relevante apenas a ordem de crescimento do tempo de execução, está sendo estudado a eficiência assintótica do algoritmo, ou seja, o estudo está preocupado com a maneira como o tempo de execução de um algoritmo aumenta com o tamanho da entrada *no limite*, à medida que o tamanho da entrada aumenta indefinidamente. Basicamente um algoritmo assintoticamente mais eficiente (ordem de crescimento mais baixa) será a melhor escolha para a maioria dos tamanhos de entrada [8].

Para o desenvolvimento desta parte foram selecionados e analisados diferentes algoritmos de ordenação a fim de avaliar o impacto das características de cada algoritmo no consumo de energia em dispositivos embarcados. Dentre os algoritmos populares os escolhidos foram: Merge Sort, Heap Sort, Counting Sort, Bucket Sort e o já mencionado anteriormente Insertion Sort em suas duas versões (iterativa e recursiva).

O conjunto de dados usados como entrada é o mesmo para todos os algoritmos. Desta forma é possível estabelecer uma analise mais precisa sobre o comportamento de cada algoritmo de ordenação. Os dados de entrada consiste em um array (lista) de 200 números inteiros positivos, onde os valores deste conjunto variam de 0 a 200.

O Merge Sort aproveita a facilidade de fundir listas já ordenadas em uma nova lista de classificação. Ele inicia subdividindo a lista inicial em listas menores sucessivamente, ordenas as subpartes e realiza o processo de fundição das listas em uma nova lista ordenada. É um algoritmo que se adapta bem a listas muito grandes pois sua ordem de crescimento no pior caso é Θ(n log n). O Merge Sort é amplamente utilizado em rotinas de classificação de linguagens de programação como Perl[10], Pyton[11] e Java[12].

O Heap Sort é uma versão muito eficiente de algoritmos de ordenação por seleção. Seu funcionamento consiste em determinar o maior elemento da lista e coloca-lo no final, continuando este processo para o restante da lista. Este processo é realizado de forma eficiente utilizando estruturas de dados [13] do tipo pilha e árvore binária. Usando uma pilha o tempo de varredura para encontrar o maior elemento é aproximadamente Θ(*log n*) ao invés de Θ(*n*) como é o caso da varredura linear nos algoritmos simples. Isso permite que o Heap Sort possa ser executado no pior caso em um tempo Θ(*n log n*).

Ao contrário dos algoritmos citados anteriormente o Counting Sort não é um algoritmo de comparação, ou seja, ele não realiza comparações entre os elementos da lista para determinar o maior elemento. O Counting Sort é aplicável quando a entrada é conhecida como pertencente a um conjunto particular S de possibilidades. O algoritmo é executado em tempo Θ(|*S*| + *n*). No entanto ele não pode ser frequentemente utilizado, pois o conjunto *S* necessita ser razoavelmente pequeno para que seja eficiente.

Por fim o algoritmo de ordenação Bucket Sort é do tipo divisão-e-conquista que generaliza o tipo contagem. Seu funcionamento consiste no particionamento de um conjunto de dados em um número finito de *buckets* ('baldes'). Cada 'balde' é então ordenado individualmente usando um algoritmo de ordenação diferente, ou recursivamente aplicando o Bucket Sort. Sua complexidade é definida como Θ(n^2) no pior caso e Θ($n + r$) no caso médio onde *r* é o número de 'baldes' em que o conjunto *n* foi particionado. O Bucket Sort não é usado em conjunto de dados que possuam uma variação muito grande, e também ele é limitado ao escopo do conjunto de dados.

IV. RESULTADOS

Todas estas análises foram realizadas em um Tablet Coby Kyros modelo MID7016 com seguintes caraterísticas: Processador ARM 11 Telechips 800 MHz, 256 MB de memória RAM e 4GB Memória Flash para armazenamento de dados e com sistema operacional Android 2.3. Após o processo de desenvolvimento e testes dos algoritmos terem sido finalizados, todos os algoritmos foram instalados no dispositivo para então iniciar-se a análise.

Os dados mostrados a seguir foram extraídos com o auxílio da ferramenta PowerTutor. A Fig. 3 ilustra a interface da ferramenta no dispositivo Android. O PowerTutor nos informa a potência em miliWatts (mW) por Segundo (S), e com estes dados é possível determinar quanto a aplicação está consumindo de energia.

Figura 3. PowerTutor em Execução.

O consumo de energia e potência são fatores limitantes na funcionalidade oferecida por alguns sistemas embarcados, principalmente os portáteis, que operam por bateria [14]. Com uma capacidade fixa de energia de uma bateria, o consumo de potência determina diretamente o tempo de vida do dispositivo embarcado.

Geralmente, os conceitos de potência e energia são confundidos. As seguintes equações (3) e (4) definem potência e energia em termos do trabalho realizado:

$$\text{Potência} = \text{Trabalho} / \text{Tempo (Watts)} \quad (3)$$

$$\text{Energia} = \text{Potência} * \text{Tempo (Joules)} \quad (4)$$

A potência utilizada por um dispositivo é a energia consumida por unidade de tempo. Já a energia é a integral da potência. Desta maneira, as baterias armazenam uma dada quantidade de energia e o objetivo é minimizar a quantidade de energia necessária para realizar cada tarefa. A energia pode ser definida pela equação (5):

$$EnergiaTotal = \int PotênciaTotal\ dt \quad (5)$$

A. Análise dos Algoritmos recursivos versus interativos de mesma complexidade

O algoritmo Insertion Sort Iterativo demostrou ter um melhor desempenho que sua versão recursiva. A partir da Fig. 4 é possível observar que a versão recursiva demorou 5 segundos a mais para executar que a versão iterativa, assim tendo aproximadamente um tempo de execução 38% maior. Com um tempo de execução menor e o pico de potência dissipada por ambas as aplicações não ultrapassarem 800 mW, a versão iterativa a partir da formula de energia apresentada anteriormente consome aproximadamente um total de 4,5 Joules (J) de energia, já a versão recursiva consome 7,9 J.

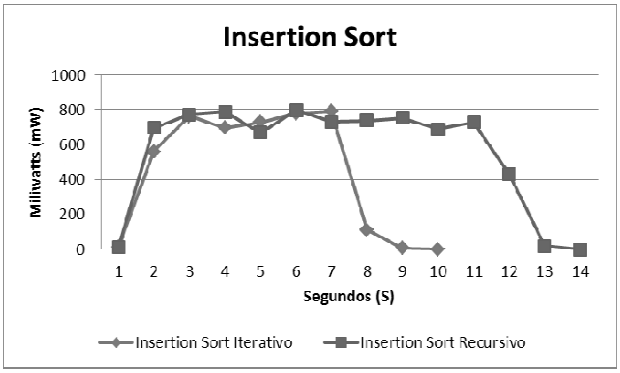

Figura 4. Gráfico Insertion Sort Iterativo e Recursivo.

Novamente a versão iterativa demostrou ter um desempenho melhor que a recursiva conforme a fig. 5, executando 20% mais rapida, assim consequentemente consumindo menos energia. A versão recursiva consumiu um total de 9,2 J, já versão iterativa consumiu 7 J.

Estes dados são facilmente explicaveis, pelo fato de algoritmos recursivos usarem intensivamente a pilha, o que requer alocações e desalocações de memória, com isso tendem a ser mais lentos que os equivalentes iterativos, isso vem a se comprovar também na plataforma Android pelas analises demostradas.

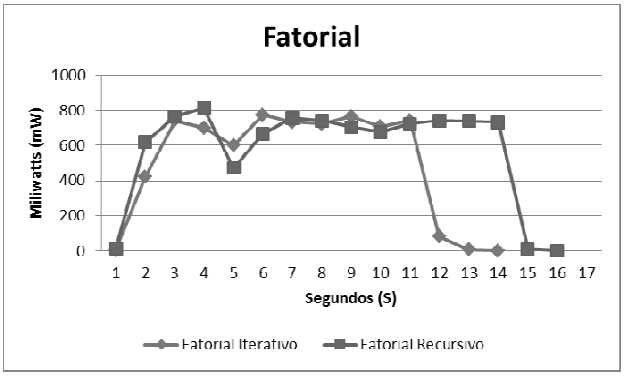

Figura 5. Gráfico Fatorial Iterativo e Recursivo.

B. Análise dos algoritmos que fazem mais uso de memória versus algoritmos que fazem mais uso de CPU

Analisando o comportamento da fig. 6, nota-se que em termos de desempenho ter uma tabela com valores já calculados e apenas acessa-los através de uma Hash é muito melhor do que ter uma computação pesada para obter esse resultado. Através da fig. 6 podemos extrair a energia consumida por ambas aplicações, na versão do seno tabelado for consumidos 1,9 J, já versão que calcula seno pela sequência, tivemos a aplicação consumindo 19 J, consumindo excessivos 90% a mais de energia.

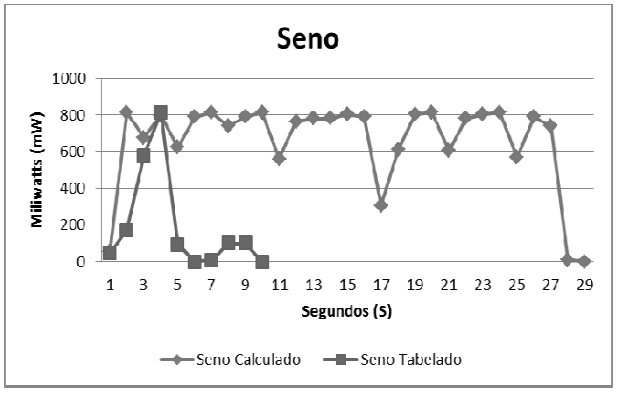

Figura 6. Gráfico do Seno Calculado e Tabelado.

Mas existe um porém, a Fig. 7 mostra que o arquivo compilado do Android (.dex) da versão tabelada ocupou 11.324 Bytes (≅ 11 Kbytes), já na versão calculada do seno o arquivo .dex ocupou apenas 4.100 Bytes (≅ 4 Kbytes). Neste caso não há muito a se recorrer, para diminuir o tempo de execução e a energia a ser consumida é necessário utilizar mais memória. Entretanto, hoje em dia isto não é um grande problema tendo em vista que a maior parte de smartphones e tablets com Android tem entre 4 GB e 16 GB de armazenamento interno, diferentemente de alguns anos atrás onde celulares tinham um espaço de armazenamento de dados bem restrito. Este tipo de memória pode ser utilizada para armazenar dados de aplicações, documentos, etc.

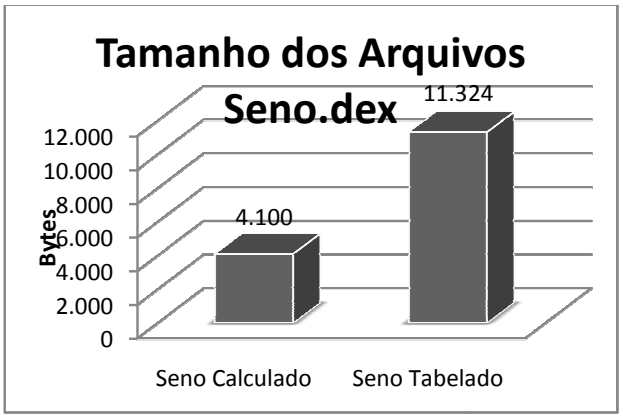

Figura 7. Gráfico do Uso de Memória do Seno Calculado e Tabelado.

C. Análise de diversos algoritmos de ordenação

Neste item serão apresentados os resultados das analises de consumo de energia dos algoritmos de ordenação. O gráfico na figura 8 apresenta uma comparação da potência dissipada pela aplicação em relação ao tempo de execução em segundos (s).

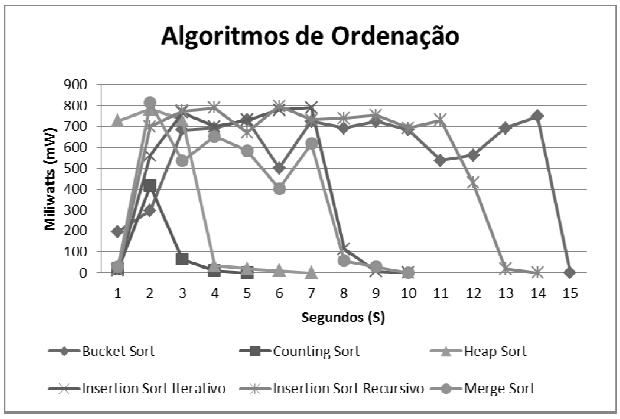

Figura 8. Algoritmos de Ordenação.

Analisando o tempo de execução de cada algoritmo, podemos perceber claramente a diferença apresentada pelo comportamento assintótico de cada um deles.

Os algoritmos de maior ordem assintótica analisados foram os que levaram mais tempo: Bucket Sort e Insertion Sort Recursivo levaram respectivamente 15s e 14s para concluir o processo de ordenação. Consequentemente, estes algoritmos consumiram mais energia: 8,5J para o Bucket Sort e 7,8J para o Insertion Sort Recursivo.

Os algoritmos Insertion Sort Iterativo e Merge Sort, obtiveram desempenho semelhante ambos levaram 10s para concluir a ordenação do conjunto de dados, e consumiram respectivamente 4,5J e 3,7J de energia. Neste caso o Merge Sort, mesmo tendo uma complexidade assintótica menor, obteve um desempenho geral praticamente igual a o Insertion Sort Iterativo, por se tratar de um método recursivo.

O Heap Sort obteve o melhor desempenho para o conjunto de entradas analisado entre os algoritmos de mesma complexidade (no caso, Merge Sort), levando 7s para concluir o processo e consumindo 2,3J de energia.

O algoritmo de ordenação Counting Sort foi o mais rápido dos algoritmos analisados. Ele levou 5s para concluir o processo e consumiu apenas 0,5J de energia. O consumo baixo de energia se deve ao fato de que o Counting Sort não utilizar de recursão e não realizar comparações durante o processo de ordenação e, consequentemente, realizado menos acesso a memória. Porém o uso deste algoritmo está limitado à condição de que os valores do conjunto de valores de entrada devem estar entre zero e o tamanho do próprio conjunto.

A figura 9 representa o gráfico do consumo de energia de cada algoritmo de ordenação analisado.

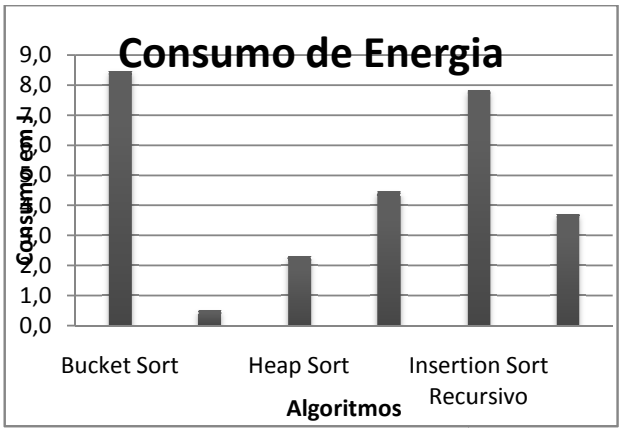

Figura 9. Consumo de Energia dos Algoritmos de Ordenação.

V. CONCLUSÕES

Este artigo apresentou uma avaliação de desempenho e consumo energético de aplicações para a plataforma móvel Android, a fim de localizar um padrão de consumo de energia em relação ao paradigma de projeto de algoritmo utilizado, assim diagnosticando a melhor versão de um determinado algoritmo para um fim específico.

Levando em conta que os dispositivos móveis mais modernos são capazes de rodar vídeo e áudio em alta definição, prover acesso à internet de alta velocidade, permitem fotografar e gravar vídeos de alta qualidade, além de possuírem interface com GPS e sensores como acelerômetro. Porém com todos esses recursos o desafio está em prolongar a vida útil da bateria, neste ponto é que se enquadra este trabalho.

As aplicações avaliadas executaram algoritmos em versões recursivas e iterativas de mesma complexidade computacional, algoritmos com foco em uso de memoria versus com foco em uso de CPU e uma analise geral de alguns algoritmos de ordenação.

Com auxilio da ferramenta PowerTutor [6] que fornece estimativas de consumo de energia por componentes de hardware, foi possível analisar o comportamento do consumo energético do dispositivo executando Android.

A partir dos dados obtidos é possível afirmar que o uso de recursividade em aparelhos com Android embarcado não é uma boa prática de programação. Em todos os algoritmos recursivos o tempo de execução foi consideravelmente maior do que algoritmos iterativos com isso consumindo mais energia. Porém a maior motivação para se escolher um algoritmo iterativo ao invés de um algoritmo recursivo é que nas plataformas Android(s) o espaço disponível para o fluxo de controle é bem menor que o espaço disponível no heap, e algoritmos recursivos tendem a necessitar de mais espaço na pilha do que algoritmos iterativos. Em todos os algoritmos recursivos quando as chamadas recursivas empilhadas chegaram perto de trezentos, houve um estouro de pilha tanto no dispositivo físico com Android embarcado, tanto no emulador disponibilizado pela Google[7].

Pela fig. 8 é visível a vantagem dos algoritmos iterativos, onde o Insertion Sort Iiterativo obteve praticamente o mesmo desempenho do Merge Sort, mesmo sendo teoricamente um algoritmo mais lento e ainda sendo superior que sua versão recursiva e do algoritmo Bucket Sort que possuem mesma complexidade.

Analisando apenas os algoritmos que não fazem uso de recursividade: Counting Sort, Heap Sort e Insertion Sort Iterativo, estes seguem o principio teórico da complexidade de cada método de ordenação, onde o algoritmo com complexidade O(n) executa em um tempo menor que um algoritmo com complexidade $O(n^2)$ e um algoritmo com complexidade O (n log n) se colocada entre o meio termo.

Com relação ao uso de memória versus o uso de CPU, os resultados obtidos demostram que se é oferecida uma boa quantidade de memória para armazenamento, uma boa solução tanto do ponto de vista de desempenho quanto de economia da bateria é substituir algoritmos que realizam calculados pesados, por uma grande tabela previamente calculada que contenha os todos ou maioria dos resultados, para quando se fizer necessário acessa-los.

A importância dos resultados obtidos com este trabalho está no fato de que conseguimos determinar o impacto de alterações de projeto de software no consumo de energia de aplicações móveis para a plataforma Android.

REFERÊNCIAS

[1] Lecheta, Ricardo R. Google Android: Aprenda a criar aplicações para dispositivos móveis com o Android SDK - 2ª edição. São Paulo: Novatec Editora, 2010.

[2] Thompson, Chris; White, Jules; Dougherty, Brian; Schmidt, Douglas C., "Optimizing Mobile Application Performance with Model–Driven Engineering".

[3] Tapas Kumar Kundu and Kolin Paul "Android on mobile devices: an energy perspective".

[4] Gartner Inc., "Gartner Highlights Key Predictions for IT Organizations and Users in 2010 and Beyond", <http://www.gartner.com/it/page.jsp?id=1278413>, acessado em: Abril de 2012.

[5] Kreiman, Edward, "Using Learning to Predict and Optimise Power Consumption in Mobile Devices".

[6] Zhang, Lide; Yang, Lei; et al. "Accurate Online Power Estimation and Automatic Battery Behavior Based Power Model Generation for Smartphones".

[7] Android, "Android Developers", <http://developer.android.com/index.html>, acessado em: Abril de 2012.

[8] Cormen, Thomas; Leiserson, Charles, e.t all, Algoritmos Teoria e Prática - 2ª edição. Rio de Janeiro: Elsevier , 2002.

[9] Lua, "The programming Lua", <http://www.lua.org/>, Acessado Maio de 2012.

[10] Perl, "Perl Programming Documentation", <http://perldoc.perl.org/functions/sort.html>, Acessado em: Julho de 2012.

[11] Python, "Sorts in Python", <http://svn.python.org/projects/python/trunk/Objects/listsort.txt>, acessado em: Julho de 2012.

[12] Java, "Java 2 Platform SE v1.3.1: Class Arrays", <http://docs.oracle.com/javase/1.3/docs/api/java/util/Arrays.html#sort>, acessado em: Julho de 2012.

[13] Tenenbaum, Aaron Ai; Langsam, Yedidyah, e.t all, Estruturas de Dados usando C - 1ª edição. São Paulo: Makron Books, 1995.

[14] F. Shearer, Power Management in Mobile Devices. Burlington, MA: Elsevier, 2008.