UNIVERSIDADE DE CAXIAS DO SUL
DEPARTAMENTO DE INFORMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RAFAEL SMIDERLE

Computação Pública para Dispositivos Móveis Baseados em *Android*

Prof. André L. Martinotto
Orientador

Caxias do Sul, Novembro de 2011

"Stay hungry. Stay foolish."
— Steve Jobs

SUMÁRIO

LISTA DE ABREVIATURAS E SIGLAS

AC	Antes de Cristo
AKS	Agrawal, Kayal e Saxena
API	*Application Programmig Interface*
ARM	*Advanced* RISC *Machine*
BOINC	*Berkeley Open Infrastructure for Network Computing*
DC	Depois de Cristo
DDR	*Double Data Rate*
GB	*Gigabyte*
GHz	*Gigahertz*
GIMPS	Great Internet Mersenne Prime Search
GMP	*GNU Multiple Precision Arithmetic Library*
GNU	*GNU's Not Unix*
HD	*Hard Disk*
IDE	*Integrated Development Environment*
iOS	*Iphone Operating System*
JNI	*Java Native Interface*
MFLOP	Megaflop
NDK	*Native Development Kit*
O	Notação Assimptótica para Complexidade de Pior Caso
PFLOP	Petaflop
PC	*Personal Computer*
PRC	*Public-Resources Computing*

RAM	*Random-Accsess Memory*
RPM	Rotações por Minuto
RSA	Rivest, Shamir e Adleman
OHA	*Open Handset Alliance*
SDK	*Software Development Kit*
SETI	*Search for Extraterrestrial Intelligence*
SO	Sistema Operacional
SVN	*Subversion*
URL	*Uniform Resource Locator*
VM	*Virtual Machine*
XML	*Extensible Markup Language*
WU	*Workunit*

LISTA DE FIGURAS

LISTA DE ALGORITMOS

RESUMO

A maior parte do poder computacional existente hoje encontra-se distribuída nos computadores pessoais e consoles de *video-game* encontrados nas casas dos usuários. Projetos de computação pública aproveitam poder computacional ocioso desses equipamentos para realizar cálculos científicos, que seriam inviáveis caso realizados em um único computador.

A popularização dos *smartphones, tablet PCs* e outros dispositivos móveis com processamento embarcado vem modelando uma nova realidade na computação doméstica. Antigamente, tarefas como navegar na internet, enviar e receber e-mails e redigir textos eram comumente executadas em um computador e raramente executadas em dispositivos móveis. Hoje percebe-se que esta realidade tem sido alterada com a popularização desses dispositivos.

O aumento do poder de processamento em tais dispositivos vem acompanhando a demanda dos usuários por aplicações cada vez mais completas e aparelhos que possam mantê-los conectados e operantes mesmo longe de um computador. Com capacidades de processamento cada vez maiores e com a sua popularização, os dispositivos móveis podem vir a apresentar uma boa contribuição para projetos de computação pública.

O objetivo deste trabalho é desenvolver um sistema de computação pública que utilize dispositivos móveis baseados na plataforma *Android* como clientes. Para a implementação do paradigma de computação pública será utilizada a plataforma BOINC, que fornece as ferramentas necessárias para a criação e execução de um projeto desse tipo.

Palavras-chave: Computação pública, BOINC, smartphone, tablet, android.

1 INTRODUÇÃO

O constante avanço tecnológico no segmento de dispositivos móveis trouxe ao alcance dos usuários *tablets* e *smartphones* cuja capacidade de processamento equivale-se à dos computadores *desktop* de uma década atrás (LONGBOTTOM, 2011). Processadores cada vez mais rápidos e sistemas operacionais (SO) sofisticados transformaram dispositivos, que antes eram simples telefones celulares, em poderosas ferramentas computacionais.

Este avanço ocasionou um aumento significativo nas vendas desses dispositivos que, por muitas vezes, acabam substituindo, além dos telefones, os computadores pessoais e até mesmo os consoles de *video-game* em tarefas como o acesso à internet, envio e recebimento de *e-mails*, edição de texto, jogos e entretenimento em geral. Essa versatilidade tem feito com que eles passem a fazer parte do dia-a-dia das pessoas e sua presença não possa ser ignorada (LECHETA, 2011).

Aliado ao aumento do poder computacional desses dispositivos, as empresas que desenvolvem os principais SOs tem disponibilizado ferramentas que facilitam cada vez mais o desenvolvimento de aplicações para dispositivos móveis. Com isso, surgiu uma infinidade de aplicativos que servem aos mais diversos fins. De calculadoras científicas e planilhas eletrônicas, como por exemplo o *Quickoffice* (QUICKOFFICE, 2011), até aplicativos que utilizam os sensores de movimento e gravidade para medir a inclinação de uma superfície, como por exemplo o *Swiss Army Knife* (TREVISTI, 2011). Assim, não é difícil imaginar a possibilidade de utilização de *softwares* para computação científica nesses dispositivos.

Um grande número de problemas científicos demandam um poder de processamento que ultrapassa os limites dos recursos computacionais disponíveis em um único computador (ANDERSON, 2004). Isso não quer dizer, necessariamente, que sua execução seja impossível. Existe uma série de alternativas para reduzir o tempo de processamento. Como exemplo, pode-se citar computação em *cluster* (BUYYA, 1999), *grid* (FOSTER; KESSELMAN, 1999) e computação pública, ou PRC (ANDERSON, 2004) que, apesar de serem modelos distintos de computação, possuem em comum a utilização de recursos computacionais distribuídos para o processamento

(SANTOS, 2005).

Se por um lado o poder de processamento dos dispositivos móveis ainda está distante dos computadores pessoais mais modernos e dos supercomputadores, por outro lado, o número elevado destes dispositivos e a tendência de crescimento do segmento podem fazer com que o uso de computação distribuída neste caso se torne uma solução viável (GARTNER, 2011) (GREENE COMPUTING, 2011a).

Dentro desse contexto, neste trabalho será desenvolvido um sistema de computação pública que possui como principal objetivo utilizar a capacidade de processamento ociosa de dispositivos móveis na busca de números primos de Fermat (COUTINHO, 2005). O alto poder de processamento exigido na busca por esse tipo de números primos deve-se à grande quantidade de algarismos dos números a serem testados, o que torna essencial a utilização de algoritmos de fatoração otimizados. Dessa forma, embora o objetivo principal do trabalho seja o desenvolvimento de um software de computação pública para dispositivos móveis, será necessário um breve estudo na área da teoria dos números que possibilite a realização destes cálculos em um tempo aceitável (COUTINHO, 2005) (RIBENBOIM, 2001).

1.1 Objetivos

O principal objetivo deste trabalho é o desenvolvimento de um sistema de computação pública para a busca de números primos de Fermat cujos clientes sejam executados sobre o SO da plataforma *Android*. Como objetivos secundários tem-se:

a) verificar a possibilidade de utilizar-se o crescente poder computacional dos dispositivos móveis atuais em sistemas de computação distribuída;

b) desenvolver um software que utilize *smartphones* e/ou *tablets* baseados na plataforma *Android* para contribuir com projetos de computação pública;

c) verificar se o desempenho oferecido hoje pelos dispositivos móveis disponíveis no mercado mostra-se satisfatório para a execução de tais sistemas.

1.2 Estrutura do Texto

Este trabalho encontra-se organizado da seguinte forma: no Capítulo 2 tem-se uma introdução ao estudo dos números primos, fatoração e sua utilização na computação. Além disso, tem-se uma breve história dos principais matemáticos que dedicaram-se ao estudo desses números, alguns métodos de verificação de primalidade e a fundamentação teórica que servirá de base para a implementação dos testes de primalidade.

O Capítulo 3 apresenta o conceito de computação pública (PRC) e um apanhado geral do funcionamento do *middleware* BOINC, que será a plataforma sobre a qual o projeto de PRC deste trabalho será desenvolvido.

O Capítulo 4 apresenta uma visão geral sobre o mercado atual de dispositivos móveis, as tendências de crescimento e o posicionamento do *Android* neste mercado. Também são abordados assuntos como a capacidade de processamento dos dispositivos móveis em comparação aos computadores pessoais (PC).

O Capítulo 5 contém os detalhes de implementação do projeto de PRC desenvolvido neste trabalho, como a instalação e configuração do servidor e o desenvolvimento da aplicação.

2 NÚMEROS PRIMOS

O estudo dos números primos, até aonde se sabe, teve início na Grécia antiga, por volta de 300 AC. A definição de número primo pode ser encontrada no livro VI dos Elementos de Euclides. Segundo ele, "um número primo é aquele que é medido apenas pela unidade". Na terminologia atual, significa que o número primo é aquele que não pode ser dividido por nenhum número menor que ele próprio exceto o número um (a "unidade" na terminologia de Euclides). Os demais números eram chamados pelos gregos de números compostos ou secundários (COUTINHO, 2004).

Ainda segundo a definição dos Elementos de Euclides, "um número composto é aquele que é medido por algum número", ou seja, aquele que é divisível por um ou mais números menores que ele próprio além do número um. Euclides dizia que qualquer número composto poderia ser obtido através da multiplicação de dois ou mais primos, porém, isso só veio a ser provado no século XIX por Carl Friedrich Gauss com a enunciação do teorema da fatoração única (COUTINHO, 2004).

2.1 Primalidade e Fatoração

O teorema da fatoração única foi descrito pela primeira vez por Gauss, no século XIX, em seu livro *Disquisitiones Arithmeticae*, porém seu conceito já era utilizado implicitamente por matemáticos desde a Grécia antiga. O teorema, que por vezes também é chamado de Teorema Fundamental da Aritmética, diz que:

Teorema 2.1.1 (Teorema da Fatoração Única) *Dado um inteiro positivo* $n >= 2$ *podemos sempre escrevê-lo, de modo único, na forma* $p_1^{e_1}...p_k^{e_k}$ *, onde* $1 < p_1 < ... < p_k$ *são números primos e* $e_1, ..., e_k$ *são inteiros positivos*

Embora saiba-se que existe uma fatoração em números primos única para qualquer número composto, isso não significa que ela possa ser encontrada em um tempo aceitável. A verdade é que os métodos de fatoração existentes são bastante ineficientes, tornando sua aplicação inviável em números muito grandes, como por exemplo, um número com 150 algarismos ou mais (COUTINHO, 2005).

O algoritmo usual de fatoração consiste em dividir o número n, o qual se deseja fatorar, por todos os números, de 2 a $\sqrt{n}$, até que se encontre um fator cujo resto da divisão seja zero. Este fator, caso exista, certamente será primo. Deve-se então repetir a operação para cada cofator obtido até que se chegue a um cofator igual a 1. Se o número não puder ser dividido por nenhum inteiro de 2 até $\sqrt{n}$ sem resto, então pode-se concluir que o número é primo (COUTINHO, 2005).

Aplicando este algoritmo para o número 450, por exemplo, teria-se o resultado em 5 iterações. Isso é algo que pode ser facilmente calculado manualmente, porém, supondo um computador que realizasse uma operação a cada microssegundo, aplicar este mesmo algoritmo para um número da ordem de 10^{100} pode demorar aproximadamente 3.10^{27} bilhões de anos (CARDOSO, 2003).

Existem outros métodos de fatoração mais eficientes do que o método das divisões por tentativas, alguns com uma alta eficiência para casos específicos. Como exemplo pode-se citar o algoritmo de Fermat, que mostra-se muito eficiente quando o número n a ser fatorado possui fatores próximos a $\sqrt{n}$. Além disso, existem testes probabilísticos altamente eficientes que identificam com alta probabilidade de acerto se um dado número é primo ou não. Como exemplo pode-se citar o teste de Miller que é utilizado, em suas variações, nos *softwares* matemáticos *Maple* (MAPLESOFT, 2011) e *Axiom* (AXIOM, 2011) dentre outros (COUTINHO, 2005).

2.2 Utilização de Números Primos na Computação

A principal aplicação dos números primos na ciência da computação encontra-se na área de criptografia de dados. A unicidade da fatoração em números primos de qualquer número composto, provada pelo Teorema da Fatoração Única de Euclides, é a base para o funcionamento do método de criptografia de chave pública mais utilizado atualmente, o RSA (RSA, 2011). O nome do algoritmo provém dos sobrenomes de seus criadores, Rivest, Shamir e Adleman (COUTINHO, 2005).

Em linhas gerais, pode-se dizer que o RSA consiste em codificar uma mensagem usando um número $n = pq$, onde p e q são números primos e a única forma de decodificar esta mensagem é saber quais são os fatores p e q que, multiplicados, dão origem ao número n. Diz-se que n é a chave pública e p e q são a chave privada. A eficácia deste sistema baseia-se em dois pontos:

a) A comprovada unicidade da fatoração de um produto de dois primos.

b) A ineficiência dos métodos atuais para fatoração de grandes números, já que fatorar n e obter p e q, quando n for um número grande (com 100 algarismos ou mais), é computacionalmente inviável (COUTINHO, 2005).

Para se ter uma ideia da importância dos números primos nesse tipo de criptografia, a *RSA Factoring Challenge* chegou a oferecer U$200.000,00 para quem encontrasse os fatores primos de um semi-primo ou pseudo-primo de 2048 *bits* (617 dígitos). Outras instituições como a *Electronic Frontier Foundation* chegaram a oferecer U$100.000,00 para quem encontrasse um número primo com 10 milhões de dígitos e até U$250.000,00 para quem encontrasse um primo de um bilhão de dígitos (ROCHA DIAS, 2008).

2.3 Classes de Números Primos

Diversos matemáticos tentaram descrever um padrão para a obtenção de números primos, que distribuem-se de maneira aleatória ao longo do conjunto dos inteiros. Embora nenhum deles tenha obtido sucesso em descrever uma fórmula para tal, muitos acabaram dando origem a classes de primos batizadas em sua homenagem. É o caso de Mersenne, Fermat, Sophie Germain e inúmeros outros. Vale ressaltar que Números de Mersenne e generalizações da fórmula de Fermat tem sido boas fontes para a obtenção de números primos de grande magnitude até hoje (ROCHA DIAS, 2008).

Uma grande vantagem em realizar a busca de números primos de forma definida como números de Mersenne e números de Fermat é que, como será descrito mais adiante, para essa classe de números existem testes de primalidade determinísticos em tempo polinomial. Isso explica o fato dos maiores primos conhecidos apresentarem esses formatos (RIBENBOIM, 2001).

2.3.1 Primos de Fermat

Pierre de Fermat, magistrado da corte de *Toulouse*, nasceu na cidade de *Beaumont-de-Lomagne*, na França, em 1601. Filho de um rico mercador de peles teve o privilégio de estudar no monastério franciscano de *Grandselve* e na Universidade de *Toulouse* sem, no entanto, ter apresentado maiores habilidades para a matemática durante seus estudos. Fermat foi o que se pode chamar de matemático amador, já que era conselheiro da corte e não fazia da matemática sua profissão principal, porém, era um estudante tão dedicado e talentoso que, ao escrever seu livro *The Mathematics Of Great Amateurs* (COOLIDGE, 1963), Julian Coolidge excluiu Fermat alegando que este "fora tão grande que deveria ser considerado profissional" (SINGH, 1997).

Com a publicação de obras gregas durante a renascença, em especial da *Aritmética de Diofanto*, Fermat passou a interessar-se pela teoria dos números. Esse interesse viria, posteriormente, expressar uma "torrente de resultados importantes". Pode-se dizer que Fermat reviveu a teoria dos números, que desde a Grécia (aproximada-

mente 250 DC) caiu "mais ou menos no esquecimento". Porém, foi Euler quem, posteriormente, popularizou a teoria dos números provando e estendendo parte dos resultados enunciados por Fermat (COUTINHO, 2005).

Dentre os resultados obtidos por Fermat, um deles enunciava que todos os números da forma $F_n = 2^{2^n} + 1$ seriam, possivelmente, primos (COUTINHO, 2005). Para $n = 0, 1, 2, 3, 4$ realmente obtém-se números primos, porém, Euler provou em 1732 que F_5 é um número composto. Até hoje os únicos primos de Fermat conhecidos são os números de F_0 a F_4 e não se sabe se existem outros. O que se sabe é que de F_5 até F_{32} todos são compostos e que F_{33} é um número de mais de 2,5 bilhões de dígitos cuja fatoração ainda é desconhecida (CALDWELL, 2011). Mesmo assim, "muitos dos maiores números primos conhecidos são generalizações dos primos de Fermat" (ROCHA DIAS, 2008, p.35). Por conveniência, ao referenciar números de Fermat neste trabalho, será utilizada a notação F_n, onde n é o expoente.

2.3.2 Primos de Mersenne

Marin Mersenne (1588-1648) foi um dos mais famosos divulgadores de resultados obtidos na matemática (GROENWALD; OLIVEIRA SAUER; FRANKE, 2005). Mersenne correspondia-se com alguns dos maiores matemáticos da época, como Descartes, Pascal e Fermat, sendo que com este último formulou os Números de Mersenne (SENA, 2008). Foi através das cartas enviadas por Mersenne que boa parte da obra de Fermat ficou conhecida, incluindo o Último Teorema de Fermat, que apenas em 1995 foi provado pelo inglês Andrew Wiles (GROENWALD; OLIVEIRA SAUER; FRANKE, 2005).

Um número primo é dito um Primo de Mersenne quando possui a forma $M_n = 2^n - 1$, sendo que n é um número primo. Dessa forma, para $n = 2, 3, 5$ e 7 tem-se números primos de Mersenne, porém, já era sabido desde a época de Mersenne que $n = 11$ gera um número de Mersenne composto e que a expressão $M_n = 2^n - 1$ gera não somente números primos, mas também números compostos. Mesmo assim, alguns dos maiores números primos conhecidos foram obtidos através dessa fórmula (ROCHA DIAS, 2008) (COUTINHO, 2005). Atualmente, o maior primo de Mersenne conhecido é $2^{43112609} - 1$, que possui 12.978.189 dígitos (CALDWELL, 2011) e foi descoberto pelo projeto GIMPS (*Great Internet Mersenne Prime Search*) (GIMPS, 2011).

O GIMPS é um projeto de computação pública iniciado em 1996 por George Woltman e que tem por objetivo a busca por números primos de Mersenne. Desde o início de suas atividades já foram descobertos 13 novos primos de Mersenne, dentre eles o maior primo conhecido, que é o número $2^{43112609} - 1$, já citado anteriormente (CALDWELL, 2011).

2.3.3 Primos de Sophie Germain

Um número primo de Sophie Germain é um número da forma $S = 2p+1$, onde p é um número primo (ROCHA DIAS, 2008). Euler e Lagrange provaram que sendo p um primo de Sophie Germain, se $p \equiv 3(mod4)$ e $p > 3$ então $2p + 1$ é primo se e somente se $2p + 1$ divide o número de Mersenne M_p (CALDWELL, 2011).

Conjectura-se existir infinitos números primos de Sophie Germain embora isso ainda não tenha sido provado (ROCHA DIAS, 2008). Atualmente, o maior primo de Sophie Germain conhecido é $183027.2^{265440} - 1$, que possui mais de 79 mil dígitos (CALDWELL, 2011).

2.4 Algoritmos para Verificação de Primalidade

Citando o artigo 329 das *Disquisitiones Aritmeticae* de Gauss, Ribenboim escreve em seu livro *Números primos: mistérios e recordes* (RIBENBOIM, 2001):

> O problema de distinguir os números primos dos números compostos e de exprimir estes últimos à custa de seus fatores primos deve ser considerado como um dos mais importantes e dos mais úteis da Aritmética... A própria dignidade da ciência requer que todos os meios possíveis sejam explorados para a resolução de um problema tão elegante e tão famoso (RIBENBOIM, 2001).

Sabendo-se da impossibilidade prática de utilizar o método das divisões consecutivas para determinar a primalidade de números grandes, surge a necessidade de se desenvolver algoritmos mais eficientes para a fatoração de números compostos e distinção de números primos. Estes algoritmos podem ser divididos em dois grandes grupos, que são: determinísticos e probabilísticos (COUTINHO, 2004). Enquanto os primeiros tem como saída um resultado exato, sempre baseado em teoremas já provados, os últimos são capazes de determinar com um certo grau de probabilidade se um número é primo ou não. Frequentemente, algoritmos probabilísticos apresentam um tempo computacional muito menor do que os algoritmos determinísticos conhecidos. A utilização de algoritmos determinísticos ou probabilísticos depende da necessidade de cada aplicação (RIBENBOIM, 2001).

Neste trabalho serão considerados apenas os algoritmos determinísticos para verificação de primalidade, já que o intuito é detectar grandes números primos para aplicações computacionais, como por exemplo, criptografia RSA. Algoritmos probabilísticos são mais comumente utilizados em softwares de computação matemática com o Maple (MAPLESOFT, 2011) e Axiom (AXIOM, 2011), onde combinados apresentam um grau de confiabilidade considerado suficiente para a aplicação (COUTINHO, 2005).

2.4.1 Crivo de Eratóstenes

Um dos mais antigos testes determinísticos de primalidade é o crivo de Eratóstenes. Ele foi proposto no século III AC. e constitui na organização dos cálculos sob a forma de um crivo. Pode-se pensar no crivo como uma espécie de peneira, que retém os números primos inferiores a um determinado valor de N (COUTINHO, 2004).

Primeiramente, o algoritmo cria uma lista com todos os valores de 2 até o número n a ser fatorado. Depois, varre esta lista desde o início removendo todos os múltiplos de 2. O próximo elemento da lista certamente será primo. Varre-se então a lista a partir deste número removendo todos os seus múltiplos e assim consecutivamente até $\sqrt{n}$. Se ao final da execução o número n ainda estiver na lista, então ele é primo. Caso contrário, ele é composto. O algoritmo do Crivo de Eratóstenes pode ser observado no Algoritmo 2.4.1.

Algoritmo 2.4.1 Crivo de Eratóstenes (SENA, 2008, p.55)

```
1: leia n
2: para i ← 2 até n faça        ▷ Cria uma lista com os valores inteiros de 2 até n
3:     vetor[i] ← i
4: fim para
5: limite ← int(√n)              ▷ Verifica-se o maior valor a ser testado.
6: para i ← 2 até limite faça
7:     para j ← i + i até limite passo i faça ▷ Remove da lista todos os múltiplos
   de i
8:         vetor[j] ← 0
9:     fim para
10:    se vetor[i] = i entao
11:        imprima: 'i é primo'  ▷ Imprime todos os primo encontrados de 2 até n
12:    fim se
13: fim para
```

Apesar de sua aplicação ser mais eficiente do que o método das divisões consecutivas, este método ainda exige um número muito grande de passos até a resolução completa do problema. Além disso, esse algoritmo exige uma elevada quantidade de memória para sua execução, que o torna impraticável para números grandes (COUTINHO, 2004) (ROCHA DIAS, 2008) (RIBENBOIM, 2001).

2.4.2 AKS (Agrawal, Kayal, Saxena)

O AKS foi desenvolvido por Agrawal, Kayal e Saxena e publicado em 2002 em um artigo intitulado *Primes in P* (AGRAWAL; KAYAL; SAXENA, 2002). O algoritmo AKS teve grande notoriedade no meio matemático, rendendo aos seus autores diversos prêmios. O que faz esse método tão especial é o fato de ser o primeiro teste de primalidade determinístico em tempo polinomial, independente de hipótese e de formato do número (SENA, 2008).

O AKS consiste em cinco etapas que podem ser observadas no Algoritmo 2.4.2. Na primeira etapa do AKS, é realizado um teste de modo a verificar se o número a ser testado é uma potência de outro número (linhas 2 a 5). Caso esse teste resulte em verdadeiro, certamente o número será composto. Na segunda etapa (linha 7), é utilizada uma equação algébrica para encontrar um valor r de forma a limitar o número de operações que serão realizadas pelo algoritmo. Na terceira etapa (linhas 9 a 18) é realizado o teste de primalidade dividindo-se o número n cuja primalidade se deseja verificar pelos números primos menores que o número r encontrado na segunda etapa. Nesta terceira etapa o algoritmo pode terminar caso encontre um divisor menor que o próprio número, retornando assim que o número é composto, ou encontrando apenas ele próprio como divisor, retornando assim que o número é primo. Caso na terceira etapa não seja definida a primalidade de n, em uma quarta etapa (linhas 20 a 25) é feita a verificação de uma série de congruências que, caso sejam verdadeiras para todo o intervalo de 1 até r, permitem afirmar que o número n é primo (COUTINHO, 2004).

Algoritmo 2.4.2 AKS (COUTINHO, 2004, p.80)

1: //Etapa 1
2: **se** n é um quadrado, cubo ou outra potência de um número inteiro **entao**
3: **imprima:** 'n é composto'
4: **pare**
5: **fim se**
6: //Etapa 2
7: Calcule $N = 2n(n-1)(n^2-1)(n^3-1)...(n^{4[log_2 n]^2}-1)$ e determine o menor número primo r que não divide N.
8: //Etapa 3
9: **se** n é divisível por algum primo $q < r$ **entao**
10: **se** $q = n$ **entao**
11: **imprima:**'n é primo'
12: **pare**
13: **fim se**
14: **se** $q < n$ **entao**
15: **imprima:** 'n é composto'
16: **pare**
17: **fim se**
18: **fim se**
19: //Etapa 4
20: **para** $b \leftarrow 1$ até r **faça**
21: **se** $(x+b)^n \neq x^n + b(mod(x^r-1))$ **entao**
22: **imprima:** 'n é composto'
23: **pare**
24: **fim se**
25: **fim para**
26: //Etapa 5
27: **imprima:** 'n é primo' ▷ Se a congruencia foi verificada para todos os valores de b

Baseado em uma generalização do Pequeno Teorema de Fermat, o custo do algoritmo AKS é $O((\log n)^{12})$ (COUTINHO, 2004). Uma vez que o objetivo é trabalhar com números muito grandes, isso representa um tempo de execução ainda alto se comparado com a maioria dos algoritmos probabilísticos, mas é um grande avanço para aplicações que exijam certeza absoluta acerca da primalidade de um número sem forma definida (ROCHA DIAS, 2008).

2.4.3 Teste de Pepin

Existe uma classe de testes de primalidade baseados em sucessões de números, onde é possível determinar a primalidade de um número n conhecendo-se a fatoração dos números $n-1$ ou $n+1$. O matemático Francois Edouard Anatole Lucas foi o iniciador desses testes, tendo desenvolvido um teste de primalidade baseado em sucessões que levou seu nome, o teste de Lucas (RIBENBOIM, 2001).

Baseado em uma generalização do teste de Lucas, o teste de Pepin é um al-

goritmo de tempo polinomial para a verificação da primalidade de um número de Fermat (COUTINHO, 2005) (POMERANCE, 1987). Por ser determinístico e possuir tempo polinomial, será o teste de primalidade adotado para o desenvolvimento deste trabalho.

De acordo com o teste de Pepin, o número de Fermat F_n é primo, para $n > 1$ se, e somente se, $5^{(F_n-1)/2} \equiv -1(mod(F_n))$ (COUTINHO, 2005). Se esta congruência mostra-se verdadeira, então o número é certamente primo, contudo, se o número for composto nenhum de seus fatores será conhecido aplicando-se o teste de Pepin (RIBENBOIM, 2001).

Algoritmo 2.4.3 Teste de Pepin (VASIGA, 2008)

```
1: leia n                          ▷ Onde n é o expoente da fórmula F_n = 2^(2^n) + 1
2: x ← 5
3: para i de 1 até 2^n − 1 faça
4:     x ← x^2(modF_n)
5: fim para
6: se x = F_n − 1 entao
7:     imprima: F_n é primo
8: senão
9:     imprima: F_n é composto
10: fim se
```

2.5 Bibliotecas para Manipulação de Números Grandes

Nas linguagens de programação mais utilizadas, como por exemplo a linguagem C, existe um limite relativamente pequeno para as variáveis do tipo inteiro. Na linguagem C, o maior número que se pode manipular com uma variável do tipo *unsigned int* de 32 *bits* é o número 4.294.967.295. Utilizando uma variável do tipo inteira de 64 *bits* esse limite é estendido para $2^{64} - 1$, um número de 20 algarismos que pode ser um limite alto o suficiente para a maioria das aplicações comerciais e até mesmo para algumas aplicações científicas, mas não para processamento de números de grande magnitude como os que serão abordados neste trabalho (HORTON, 2006).

O número de Fermat F_6 possui 20 algarismos (WOLFRAM, 2011) e já extrapola o limite de uma variável de 64 *bits* sem sinal (HORTON, 2006). Já F_{33} é um número de mais de 2,5 bilhões de algarismos (CALDWELL, 2011). Sendo assim, a busca por números primos de Fermat torna obrigatória a utilização de bibliotecas específicas para a manipulação de grandes números. Dentre as alternativas disponíveis estão a GMP (GNU *Multiple Precision Arithmetic Library*) (GMP, 2011), que pode ser utilizada com as linguagens de programação C ou C++ (GMP, 2011) e a *BigInteger*, que acompanha o Java (TABORDA, 2011).

2.5.1 GMP - GNU *Multiple Precision Arithmetic Library*

Criada em 1991 para a manipulação de números grandes, a GMP é uma biblioteca que permite trabalhar com números de grande magnitude, sendo a quantidade de algarismos limitada apenas pela quantidade de memória disponível. A GMP implementa um grande número de funções matemáticas, incluindo funções de teoria dos números e foi construída para ser o mais rápida possível (PIZZAMIGLIO, 2005). Por ser de fácil utilização e possui uma boa documentação em seu *website (http://gmplib.org)*, a GMP foi a biblioteca utilizada para manipulação de números grandes neste trabalho.

Inicialmente, foi desenvolvido um programa em linguagem de programação C utilizando a biblioteca GMP com o objetivo de demonstrar a eficácia do teste de Pepin na identificação de números primos de Fermat e a possibilidade de utilizar a GMP para computação de grandes números. No Algoritmo 2.5.1, pode ser observada a implementação do teste de Pepin utilizando a GMP.

Algoritmo 2.5.1 Implementação do Teste de Pepin utilizando a GMP

```
1: #include <stdio.h>
2: #include <stdlib.h>
3: #include <math.h>
4: #include <gmp.h>
5: int pepin(double k, mpz_t fn){
6:     mpz_t x,fk;
7:     mpz_init(x);
8:     mpz_init(fk);
9:     mpz_set_ui(x,5);
10:    long double i, n;
11:    n = pow(2,k) -1;
12:    for(i=1;i<=n;i++){
13:        mpz_powm_ui(x,x,2,fn);
14:    }
15:    mpz_sub_ui(fk,fn,1);
16:    if (mpz_cmp(x,fk) == 0 || k == 0 || k == 1){
17:        mpz_clear(x);
18:        mpz_clear(fk);
19:        return(1);
20:    }
21:    else{
22:        mpz_clear(x);
23:        mpz_clear(fk);
24:        return(0);
25:    }
26: }
```

```
27: int main(){
28:     mpz_t fn;
29:     mpz_init(fn);
30:     int i;
31:     for (i = 0; i<30;i++){
32:         long expoente = pow(2,i);
33:         mpz_ui_pow_ui(fn,2,expoente);
34:         mpz_add_ui(fn,fn,1);
35:         if(pepin(i,fn)){
36:             if(mpz_sizeinbase(fn,10) > 100) {
37:                 printf("F_%d eh primo de fermat e possui %d digitos \n", i,
38:                         mpz_sizeinbase(fn,10) );
39:             }
40:             else {
41:                 printf("F_%d = %s eh primo de fermat \n", i,
42:                         mpz_get_str(NULL,10,fn));
43:             }
44:         }
45:         else {
46:             if(mpz_sizeinbase(fn,10) > 100) {
47:               printf("F_%d NAO eh primo de fermat e possui %d digitos \n", i,
48:                         mpz_sizeinbase(fn,10) );
49:             }
50:             else{
51:                 printf("F_%d = %s NAO eh primo de fermat \n", i,
52:                         mpz_get_str(NULL,10,fn));
53:             }
54:         }
55:     }
56:     mpz_clear(fn);
57: }
```

Nas linhas 6 e 28 tem-se a definição de variáveis do tipo *mpz_t*. Esse tipo de variável é disponibilizado pela GMP e é utilizada para manipular números inteiros de grande magnitude. Como pode ser observado nas linhas 7, 8 e 29, é necessário utilizar a função *mpz_init(mpz_t)* para inicializar essas variáveis antes de utilizá-las. Também é recomendado utilizar a função *mpz_clear(mpz_t)* (linhas 17, 18, 22, 23 e 56) para limpar o conteúdo dessas variáveis ao final de sua utilização, evitando assim um consumo desnecessário de memória.

Na linha 9, a função *mpz_set_ui(mpz_t,long int)* é utilizada para atribuir um valor numérico do tipo *long int* para uma variável do tipo *mpz_t*. Já na linha 13, é atribuído à variável x o resto da divisão inteira entre x^2 e fn. A função *mpz_powm_ui(mpz_t,mpz_t,long int,mpz_t)* eleva o número passado no segundo parâmetro a uma potência definida no terceiro parâmetro e atribui à variável informada no pri-

meiro parâmetro o resto da divisão inteira entre o resultado desta exponenciação e o número passado no último parâmetro da função.

Na linha 15, realiza-se a subtração de uma unidade do valor contido em fn através da função *mpz_sub_ui(mpz_t,mpz_t,long int)*, atribuindo o resultado da subtração à variável fk, que é informada no primeiro parâmetro desta função.

Na linha 16, utiliza-se a função *mpz_cmp(mpz_t, mpz_t)* para realizar a comparação de valores entre duas variáveis do tipo *mpz_t*. Caso as duas variáveis possuam valores iguais essa função retorna zero. Caso possuam valores diferentes, então ela retorna 1 se o valor da primeira variável for maior que o da segunda e -1 se o valor da primeira variável for menor que o da segunda.

Finalmente, a função *mpz_sizeinbase(mpz_t,int)* é utilizada nas linhas 36, 38, 46 e 48 para obter-se o número de algarismos de uma variável do tipo *mpz_t*, onde o segundo parâmetro indica a base em que este valor será representado.

2.5.2 Tempo de Execução do Teste de Pepin

O programa que executa o teste de Pepin, implementado com o uso da biblioteca GMP e descrito na Seção 2.5.1, foi executado em um computador com processador de núcleo duplo Intel *Core* 2 *Duo* U7300, sendo que cada núcleo possui um *clock* de 1,3GHz. Além disso esse computador possui 4GB de memória RAM DDRII, um HD de 320GB com rotação de 5400 RPM e executa um sistema operacional *Linux Ubuntu* com *kernel* 2.6.35-22. Os tempos de execução medidos podem ser observados no gráfico da Figura 2.1.

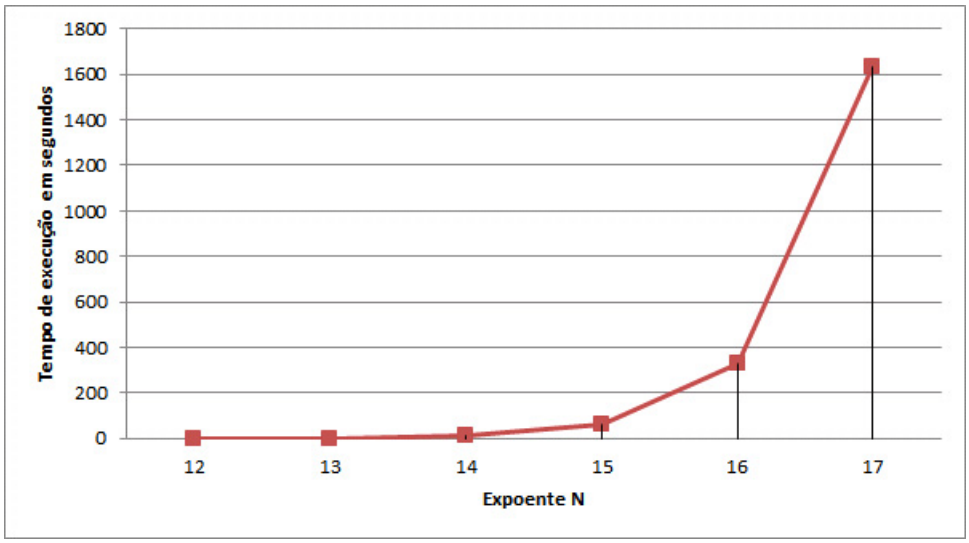

Figura 2.1: Tempos de Execução do Teste de Pepin

Embora o teste de Pepin seja considerado um teste de custo polinomial (RIBENBOIM, 2001), analisando o gráfico nota-se uma taxa de crescimento exponencial no tempo de processamento em função do expoente N utilizado. Isso ocorre pois, ape-

sar do algoritmo apresentar um tempo de computação polinomial em função do tamanho do número de Fermat a ser verificado (RIBENBOIM, 2001), a forma duplamente exponencial dos números de Fermat (COUTINHO, 2005) faz com que o número cresça muito rapidamente em função do expoente, tornando a computação de grandes números de Fermat um processo altamente custoso.

3 COMPUTAÇÃO PÚBLICA

A maior parte do poder computacional e da capacidade de armazenamento disponíveis não encontra-se centralizada em super-computadores nem em *clusters* ou *grids*. Ao invés disso, a maior parte dos recursos computacionais disponíveis encontra-se distribuída nos computadores pessoais e consoles de *video-game* encontrados nas casas das pessoas (ANDERSON, 2004).

A computação pública, também chamada de computação voluntária ou PRC (*Public-Resources Computing*), é uma forma de computação que "utiliza computadores conectados à internet, cedidos pelos seus proprietários, como fonte de poder computacional e armazenamento" (ANDERSON; FEDAK, 2006). Projetos de computação pública não uma são novidade, e consistem, essencialmente, em utilizar o tempo ocioso dos computadores de voluntários (através de um *software* cliente neles instalado) para executar cálculos científicos. Os resultados destes cálculos são então enviados a um servidor para que sejam analisados.

Essa forma de computação emergiu em meados da década de 1990 com projetos como o GIMPS e o *Distributed.net* (DISTRIBUTED, 2011), destacando-se principalmente o projeto SETI@*Home*, lançado em 1999 e que conta hoje com a surpreendente capacidade de 3.5 PFLOP distribuídos em mais de 5 milhões de computadores participantes ao redor do mundo (ANDERSON, 2004) (KORPELA et al., 2011).

3.1 Principais Projetos de PRC

O modelo PRC vem sendo utilizado em vários projetos como, por exemplo, os já citados GIMPS (GIMPS, 2011) e SETI@*Home* (SETI@HOME, 2011), o *Folding@Home* (FOLDING@HOME, 2011) , o *Climate Prediction.net* (PREDICTION, 2011) e tantos outros. Entre esses, merece especial destaque o *World Community Grid*, criado e apoiado pela IBM, e que tem a missão de "ser o maior *grid* de computação pública do mundo para apoiar projetos em benefício da humanidade" (WORLD COMMUNITY GRID, 2011).

Existem ainda projetos inovadores como o Foldit (FOLDIT, 2011), que ao invés de executar cálculos em segundo plano como a maioria dos projetos de PRC, tenta utilizar a habilidade de reconhecimento de padrões do ser humano para descobrir novos formatos de proteínas. O projeto consiste em um jogo em que o usuário é desafiado a reconhecer e montar padrões de enovelamento de proteínas na forma de quebra-cabeças como pode-se ver na Figura 3.1 (FOLDIT, 2011). Recentemente esse projeto anunciou a descoberta de "um formato de proteína que nenhum cientista ou computador conseguiu nos últimos dez anos." (TECNOBLOG, 2011). Ao competir por pontos na resolução de quebra-cabeças os jogadores acabaram descobrindo a estrutura proteica de uma enzima responsável por ajudar na reprodução de retrovírus como, por exemplo, o HIV (TECNOBLOG, 2011).

Figura 3.1: Proteína desdobrada no *Foldit*

O sucesso de um projeto de computação pública depende de muitos fatores. Entre eles, os mais importantes são o número de computadores disponibilizados pelos voluntários, o tempo de computação cedido por cada um e o poder de processamento desses computadores. Um fator importante para a adesão e fidelização de voluntários a esse tipo de projeto é a forma como seus participantes são recompensados. Entre as formas adotadas, as mais comuns são: pontos por participação, participação em sorteios e o retorno regular da importância de sua participação no projeto e dos resultados alcançados (TAYLOR, 2006) (NOV; ANDERSON; ARAZY, 2010).

3.2 Plataforma BOINC

O BOINC (*Berkeley Open Infrastructure for Network Computing*) é um *middleware* desenvolvido na Universidade de *Berkeley* que visa facilitar a criação de sistemas de computação pública. Esse *middleware* reúne toda a infraestrutura ne-

cessária para o desenvolvimento desse tipo de sistema na forma de um *framework*. O BOINC foi desenvolvido pela mesma equipe do SETI@*Home* quando o processamento disponível ao projeto ultrapassou o necessário. Dessa forma, eles optaram por criar uma ferramenta que permitisse que outros desenvolvedores utilizassem esse paradigma de computação e os recursos computacionais disponíveis para diminuir o tempo de processamento em seus projetos (SANTOS, 2005).

A estrutura de funcionamento do BOINC pode ser dividida, basicamente, em três componentes: servidor, cliente e aplicação. Uma visão geral da arquitetura BOINC pode ser encontrada na Figura 3.2. No BOINC, o cálculo a ser realizado é dividido em diversas tarefas menores (nomeadas *workunits* - WU), sendo que cada tarefa é enviada para um ou mais clientes para que seja processada. O resultado do processamento é então enviado novamente ao servidor para a análise (SANTOS, 2005).

O BOINC possui mecanismos para o controle da pontuação dos usuários, que podem ser utilizados como forma de recompensa para aqueles que participam mais ativamente do projeto. Para evitar fraudes no sistema de pontuação, prevenindo que resultados incorretos sejam enviados pelos participantes apenas para acumular pontos, cada WU pode ser enviada para mais de um cliente e o servidor realiza a validação dos resultados antes de creditar os pontos (SOTTRUP; PEDERSEN, 2005).

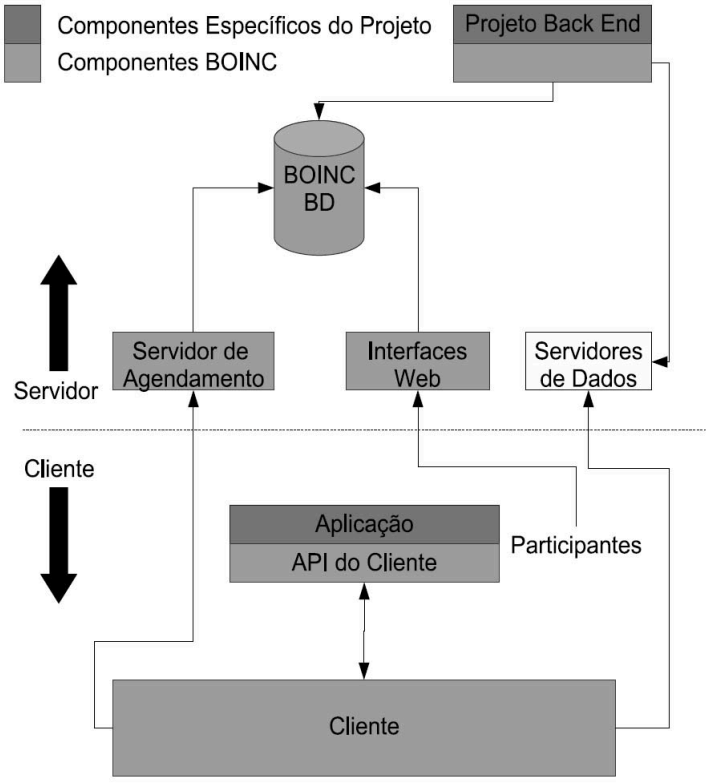

Figura 3.2: Arquitetura BOINC

Por apresentar uma solução bastante completa e estável, uma vez que é utilizado em grandes projetos de PRC, o BOINC foi a ferramenta escolhida para a criação e gerenciamento do projeto de PRC desenvolvido neste trabalho. Alguns detalhes do seu funcionamento serão detalhados nas próximas seções.

3.2.1 Servidor BOINC

O servidor BOINC é composto por um banco de dados onde são armazenadas as informações de cada projeto tais como usuários participantes, a relação das WUs disponíveis, enviadas e processadas bem como o estado de cada uma delas; por um servidor *Web* para interação com os participantes e distribuição das WUs; por um servidor de dados para armazenamento das WUs e por um servidor de agendamento com o qual o cliente irá comunicar-se para solicitar novas WUs. Na Figura 3.3 tem-se a estrutura do servidor BOINC (SANTOS, 2005).

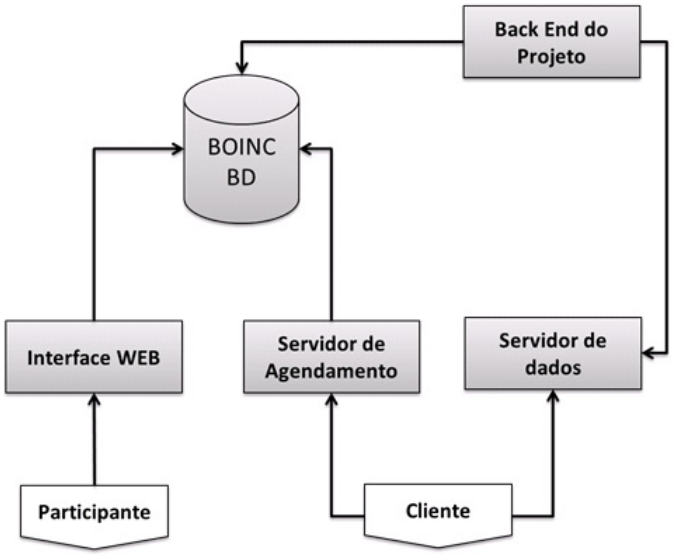

Figura 3.3: Servidor BOINC

Além disso, o servidor conta com mais cinco processos, que são: o transacionador, o validador, o assimilador, o *file deleter* e o escalonador. Esses processos são responsáveis, respectivamente, por controlar o estado transacional de cada WU, realizar a validação e assimilação dos resultados enviados pelos clientes, deletar as WUs já assimiladas e controlar a prioridade de envio das WUs que ainda não foram processadas (SOTTRUP; PEDERSEN, 2005). O validador e o assimilador são parte do *back-end* do projeto, ou seja, realizam o processamento específico de cada projeto no servidor. Esses dois programas devem ser implementados pelo administrador do sistema de forma que atendam as necessidades individuais de cada projeto (SANTOS, 2005).

O validador é responsável por garantir que os resultados enviados pelos clientes

sejam válidos. Um resultado inválido pode ser fruto de uma fraude, um erro no cliente ou mesmo na aplicação. Portanto, assim que um resultado da WU é recebido, o validador seta o estado dessa WU como "precisa ser validada". Após, ele compara todos os resultados de uma WU que necessita ser validada de forma a garantir que exista um número mínimo de resultados iguais, ou seja, um resultado canônico para esta WU. O número de resultados iguais para que se possa considerar um resultado canônico é definido pelo administrador do sistema (SANTOS, 2005) (SOTTRUP; PEDERSEN, 2005).

O assimilador é específico de cada projeto e é o responsável por realizar qualquer operação que seja necessária sobre os resultados de cada WU. O BOINC fornece dois modelos de assimilador, um em linguagem de programação C e outro em *Python*, sendo que em ambos existe uma função *assimilate_handler* que deve ser implementada pelo administrador do projeto. É nessa função que deve ser implementada a operação que se deseja realizar sobre cada resultado parcial, que pode ser, por exemplo, multiplicar o seu valor pelo produto dos resultados anteriores e armazenar esse novo valor em uma fila para aguardar a conclusão da próxima WU. Após a execução do assimilador, o estado "finalizado" é atribuído à WU, podendo essa ser deletada pelo *file deleter* (SANTOS, 2005).

Neste trabalho, foi utilizada a estrutura de servidor do BOINC para realizar o gerenciamento dos usuários bem como todo o controle de distribuição de WUs e assimilação dos resultados.

3.2.2 Cliente BOINC

O cliente BOINC, é o aplicativo executado no computador voluntário. Os projetos de PRC costumam disponibilizar o cliente BOINC padrão para *download* na página do projeto. Esse *download* também pode ser realizado no *website* oficial do BOINC *(http://boinc.berkeley.edu/)*. O cliente BOINC realiza a autenticação do usuário no projeto, o *download* da aplicação para execução dos cálculos específicos do projeto e das WUs.

O cliente BOINC padrão também proporciona toda a interface com o usuário e realiza o controle sobre a utilização dos recursos do computador do voluntário, permitindo que o usuário defina o percentual de uso do processamento, memória e armazenamento em disco que serão utilizados para os projetos de computação pública gerenciados pelo cliente. É possível também definir o percentual de tempo que será destinado a cada projeto, uma vez que um cliente pode gerenciar inúmeros projetos simultaneamente (SOTTRUP; PEDERSEN, 2005).

O cliente BOINC padrão pode gerenciar inúmeros projetos, sendo que para cada projeto será realizado o *download* da sua aplicação específica para o processamento de suas WUs (SOTTRUP; PEDERSEN, 2005). Uma representação da arquitetura

do cliente pode ser encontrada na Figura 3.4.

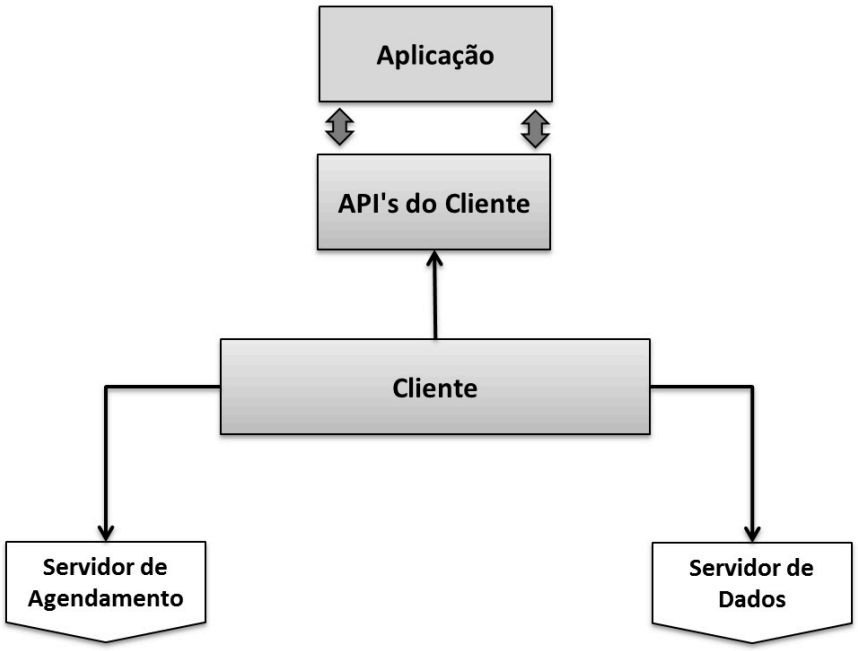

Figura 3.4: Cliente BOINC

Os esforços para se executar o cliente BOINC sobre plataformas móveis iniciaram em 2008, quando uma equipe liderada por Oded Ben-Dov portou o projeto SETI@*Home* e uma versão simplificada do cliente BOINC para a linguagem de programação JAVA na plataforma *Android* (BOINCOID, 2012) (BOINC, 2012a). Essa foi uma prova de conceito, uma vez que o cliente desenvolvido não poderia executar outros projetos exceto o SETI@*Home*. Além disso, reescrever o cliente BOINC na linguagem JAVA não é mais uma abordagem interessante, uma vez que atualmente é possível utilizar o NDK (*Native Development Kit*) para compilar programas em C de forma nativa para o *Android* (BOINC, 2012a).

Posteriormente - em 2009 - Michael Black, um professor de Ciência da Computação na *American University*, desenvolveu um protótipo em *Python* intitulado "BOINC *on iPhone* ®" que podia ser executado sobre o iOS (BOINC, 2012a).

Já no ano de 2011 o projeto *AndroBOINC*, desenvolvido por Pavol Michalec, permitiu o controle remoto de clientes BOINC através de uma interface gráfica em dispositivos que executam *Android*. Este projeto ainda não tornava o dispositivo móvel um cliente em si, mas possibilitava o controle remoto de outros clientes e abria a possibilidade para que, futuramente, se controlasse um cliente executado localmente (BOINC, 2012a).

Finalmente, em março de 2012, desenvolvedores do projeto *Native* BOINC *for Android* conseguiram compilar uma versão modificada do cliente BOINC para *Android*. A partir de então, foi possível utilizar o *AndroBOINC* como gerenciador

desse cliente. Essa solução permitiu que os projetos já existentes pudessem contar também com clientes da plataforma *Android* (BOINC, 2012a) (NATIVE, 2012).

Dessa forma, neste trabalho optou-se pela utilização do *Native* BOINC *for Android* para a computação das WUs nos dispositivos móveis. Entretanto, para sua utilização, é necessário ainda que a aplicação seja compilada para a arquitetura ARM (*Advanced* RISC *Machine*) utilizando as bibliotecas modificadas do cliente BOINC para *Android*. É necessário, também, adicionar a plataforma "arm-android-linux-gnu" ao servidor. Esses procedimentos foram necessários para o funcionamento do projeto desenvolvido neste trabalho e serão detalhados no Capítulo 5.

3.2.3 Aplicação

A aplicação é quem efetivamente realiza o processamento do projeto. A aplicação é composta por um programa que é executado através do cliente BOINC e tem como objetivo o processamento de WUs, por um validador responsável por impedir fraudes nos resultados e por um assimilador, que é executado no servidor e tem como objetivo interpretar os resultados das WUs para formar um resultado final.

Quem controla a execução da aplicação no computador voluntário é o cliente BOINC através de um conjunto de APIs. Isso possibilita que o cliente BOINC controle os recursos utilizados pela execução da aplicação e realize *checkpoints* periódicos, salvando o estado de memória do computador voluntário para que o processamento possa ser interrompido e continuado posteriormente (SOTTRUP; PEDERSEN, 2005). As principais vantagens da utilização dessas APIs neste trabalho são:

a) controle da execução pelo cliente BOINC para que a aplicação seja executada apenas enquanto o dispositivo estiver conectado ao carregador de bateria ou conforme as preferências do usuário.

b) acompanhamento do progresso de cada WU em tempo real, realizando o envio dos resultados para o servidor do projeto automaticamente ao fim do processamento.

c) realização de *checkpoints*, permitindo que o processamento seja interrompido e retomado a qualquer momento sem que haja uma perda significativa de tempo de processamento.

Além disso, através das APIs fornecidas pelo cliente BOINC a aplicação pode renderizar gráficos em tela e prover uma proteção de tela animada, tornando o projeto mais atrativo para o usuário (ANDERSON, 2004). É recomendável que a aplicação utilize estas APIs para a geração de gráficos de acompanhamento do andamento das WUs e do projeto como um todo. Embora a riqueza gráfica e a forma como a aplicação retorna ao usuário as informações sobre o andamento do

projeto sejam fatores que podem influenciar no sucesso deste (NOV; ANDERSON; ARAZY, 2010), a aplicação que será desenvolvida não fará uso de recursos gráficos como proteção de tela ou animações. A utilização desses recursos foge ao propósito deste trabalho, mas poderá ser abordada por trabalhos futuros.

Além de utilizar a API BOINC, a aplicação desenvolvida neste projeto será implementada em linguagem de programação C, utilizando a biblioteca GMP para manipulação de números grandes.

4 COMPUTAÇÃO MÓVEL

Desde o lançamento da primeira rede de telefonia celular, em 1979, no Japão, o segmento de dispositivos móveis tem apresentado um crescimento acelerado. Telefones celulares, que àquela época eram acessórios equipando automóveis de luxo, tornaram-se menores, mais eficientes e mais acessíveis nas décadas que se passaram (MATEUS; LOUREIRO, 2005).

Na década de 90 os telefones celulares popularizaram-se e ganharam novas funcionalidades. Os chamados *smartphones*, telefones com capacidade de executar aplicativos e acessar a internet, tornaram-se parte do dia-a-dia das pessoas e, com a redução dos custos para a transmissão de dados, passaram a substituir o computador pessoal em muitas atividades (MATEUS; LOUREIRO, 2005).

Dados recentes do instituto Gartner indicam o crescimento de 74% na venda de *smartphones* no segundo trimestre de 2011 em comparação com o mesmo período de 2010. Os últimos estudos do instituto Gartner apontam que, os *smartphones* já representam 31% do total de dispositivos móveis vendidos (GARTNER, 2012) e dados da consultoria IDC apontam que o volume global de vendas desses aparelhos ultrapassará o dos PCs já neste ano de 2012 (NASCIMENTO RIBEIRO, 2011). Assim, com a possibilidade de utilização desses dispositivos em computação pública, somente no segundo trimestre de 2011 ter-se-ia um total de 107,2 milhões de novos voluntários potenciais.

No quesito performance, alguns *smartphones* atuais contam com processadores de múltiplos núcleos e até aceleração gráfica. Em testes de cálculo com ponto flutuante, realizados através do *benchmark* Linpack para *Android* (GREENE COMPUTING, 2011b), esses dispositivos chegaram a atingir marcas superiores a 250 MFLOP (GREENE COMPUTING, 2011a). Esse desempenho é comparável ao de um processador Intel Duron 600 da geração Pentium III ou ainda ao moderno Intel Atom 455, que equipa modelos de *netbooks* atuais (LONGBOTTOM, 2011).

4.1 Plataformas

Assim como os computadores, os *smartphones* executam um sistema operacional que realiza o gerenciamento dos recursos de *hardware*, controla a execução de programas e disponibiliza uma interface para interação com o usuário. Existem diversos sistemas operacionais para dispositivos móveis, sendo que os sistemas com maior participação no mercado são, na ordem; *Android*, *Symbian* e *iOS* (GARTNER, 2011).

Nas próximas seções, será realizado um comparativo entre os três sistemas operacionais para dispositivos móveis com maior participação no mercado. Esse comparativo será realizado com foco na aplicação que será desenvolvida neste trabalho.

4.1.1 Android

O *Android* é a resposta da *Google* para atender ao mercado de dispositivos móveis (LECHETA, 2011). Esse, foi desenvolvido em uma parceria da *Google* com 34 companhias de mobilidade e tecnologia, a *Open Handset Alliance* (OHA). O *Android* não é apenas um sistema operacional, mas "um conjunto de softwares para dispositivos móveis que inclui sistema operacional, *middleware* e aplicativos chave" (CUCCARO, 2008).

O *Android* é a "primeira plataforma para aplicações móveis completamente livre e de código aberto". O SO é baseado em *Linux*, mais especificamente na versão 2.6 do *kernel*, e sua licença flexível permite que os fabricantes possam realizar customizações no código-fonte sem a necessidade de compartilhar essas alterações (LECHETA, 2011).

A linguagem de programação preferencial para o desenvolvimento de programas para a plataforma é o JAVA, embora seja possível desenvolver aplicativos em C e C++ (DEVELOPERS, 2011), e o ambiente de desenvolvimento adotado pela *Google* é o *Eclipse* (ECLIPSE ORG, 2011). Esse ambiente já é bastante difundido entre os desenvolvedores e é disponibilizado para diversas plataformas (NASCIMENTO RIBEIRO, 2011).

No *Android*, não há uma máquina virtual JAVA, mas sim, o *Dalvik Virtual Machine*, que executa programas no formato *Dalvik* executável (.dex). Este arquivo é gerado pela ferramente "dx", incluída no SDK *Android*, a partir dos *byte codes* gerados pelo compilador JAVA e é otimizado para o baixo consumo de memória (NASCIMENTO RIBEIRO, 2011).

Como pode ser observado na Figura 4.1, todas as aplicações, assim como o *runtime* da máquina virtual *Dalvik*, são executadas em uma *sandbox*, que é o termo utilizado para designar uma camada com restrições de execução, onde a aplicação possui permissões limitadas. Isso significa que as aplicações tem acesso limitado aos

diretórios do sistema e dispõem apenas de um conjunto de APIs para acessar os recursos do SO (MOBILTEC, 2011).

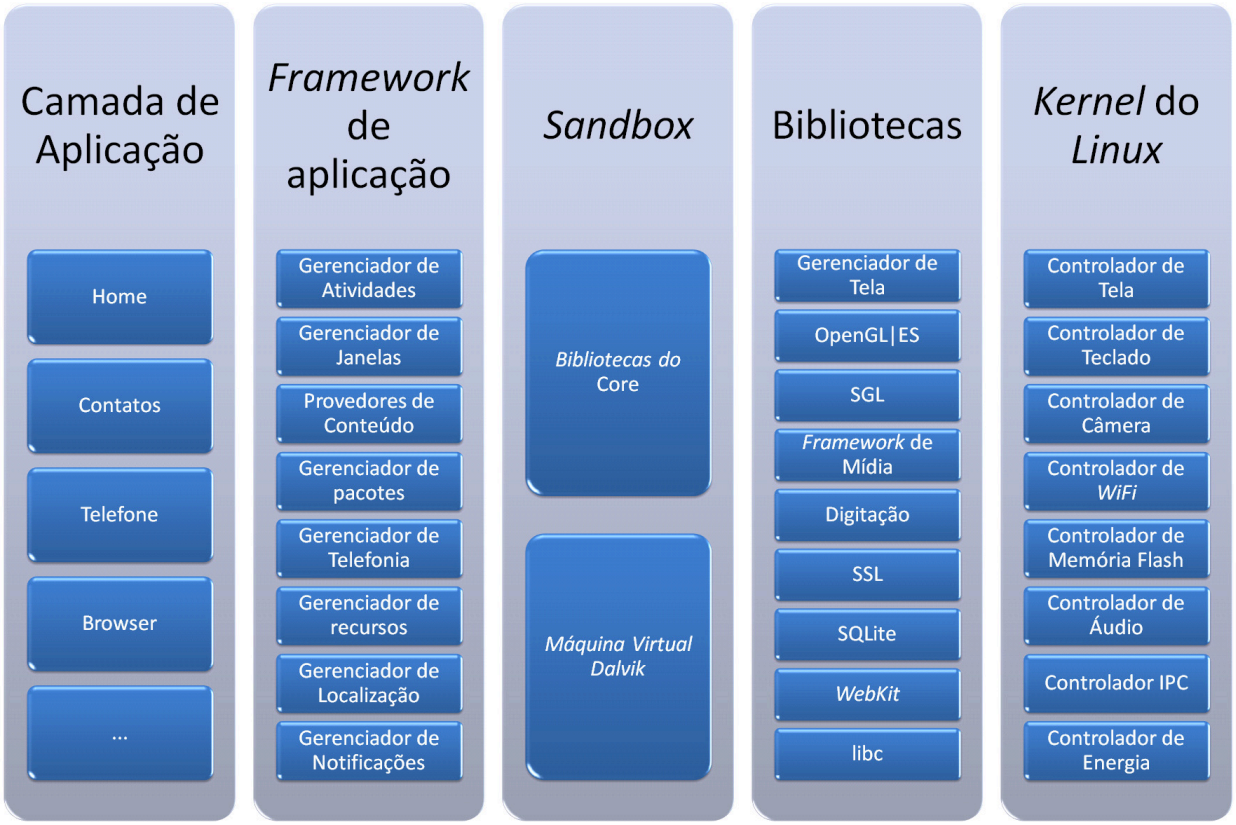

Figura 4.1: Arquitetura *Android* (MOBILTEC, 2011)

4.1.2 *iPhone* OS

O iPhone OS, ou iOS, é o sistema operacional da *Apple* para dispositivos móveis que equipa o *iPhone*, o *iPad* e o *iPod Touch*. Seu *kernel* é baseado no *kernel* do Mac OS X, portanto, seu código-fonte é proprietário e não pode ser modificado livremente assim como ocorre com o *Android* (NASCIMENTO RIBEIRO, 2011).

A linguagem de programação nativa do *iOS* para o desenvolvimento dos aplicativos é *Objective-C*, mas também podem ser utilizadas as linguagens C e C++. O único ambiente de desenvolvimento disponível é o *XCode*, que oferece ferramentas superiores às do SDK do *Android*, porém, o fato desse ser disponibilizado somente para o sistema operacional *Mac OS* restringe e encarece o processo de desenvolvimento de aplicativos (NASCIMENTO RIBEIRO, 2011).

Os programas desenvolvidos em *Objective-C* são compilados pelo próprio SDK, gerando um executável nativo para o *iOS*. Porém, embora o *iOS* não utilize uma máquina virtual, não significa que os programas poderão acessar diretamente os recursos de *hardware* e arquivos do sistema operacional. Assim como no *Android*, os programas são executados em uma *sandbox* de forma a garantir uma maior segurança do sistema (MOBILTEC, 2011).

4.1.3 Symbian

O *Symbian* é um sistema operacional para dispositivos móveis, criado em torno do ano 2000, fruto de um empreendimento conjunto entre a empresa *Psion* e as empresas líderes de mercado na telefonia móvel como *Nokia*, *Ericson*, *Motorola* e *Matsushita* (*Panasonic*). Esse empreendimento conjunto assumiu a propriedade do sistema operacional EPOC, já utilizado pela *Psion* em seus dispositivos móveis, dando origem ao *Symbian* versão 6. O sistema foi assim chamado pois o sistema EPOC já estava na versão ER5 (TANENBAUM, 2010).

O *Symbian* é um sistema operacional de micronúcleo (*microkernel*). Isso significa que funções consideradas não essenciais foram removidas do *kernel* do sistema operacional, sendo acessadas através de servidores de recursos (TANENBAUM, 2010).

A vantagem desse tipo de arquitetura é que, geralmente, sistemas de *microkernel* consomem menos memória e possuem uma inicialização dinâmica. Os recursos podem ser acoplados ao núcleo na forma de módulos sob demanda, sem que seja necessário carregá-los na inicialização do sistema operacional. Isso sem dúvida representa uma grande vantagem para dispositivos móveis com capacidades limitadas como *smartphones* (TANENBAUM, 2010).

Apesar do *Symbian* ser dito um SO aberto, ao menos na versão 6, não significa necessariamente que o seu código-fonte o seja. O *Symbian* é aberto no sentido em que a estrutura do sistema operacional foi publicada e as interfaces estão disponíveis para os desenvolvedores, incentivando o desenvolvimento de *softwares* de terceiros (TANENBAUM, 2010).

Assim como ocorre no *iOS* e no *Android*, as aplicações para o *Symbian* podem ser desenvolvidas utilizando-se, dentre outras, as linguagens de programação C, C++ e JAVA (VALENTE, 2007). Existe um SDK base para o desenvolvimento de aplicações no *Symbian*, porém, os problemas começam pela segmentação do próprio SDK. Aplicativos desenvolvidos utilizando o SDK de um fabricante podem não funcionar em aparelhos de outro. Isso deve-se ao fato de que nem todos os SDK implementam todas as funções do SDK base. Além disso, o SDK também é atrelado a uma versão específica do sistema operacional, o que significa que aplicações desenvolvidas com o SDK da versão 6 possivelmente não funcionarão em versões posteriores (DAMASIO, 2006).

O *Symbian* também deixa de ser uma opção interessante para o desenvolvimento de projetos de PRC a partir do momento em que suas vendas encontram-se em queda (GARTNER, 2011) e, principalmente, pelo fato de que o principal fabricante de dispositivos móveis que utilizam esse SO anunciou que vai abandoná-lo nos seus próximos *smartphones* (IDG NOW, 2011).

Por fim, devido à segurança do sistema *Symbian*, dependendo dos recursos que o *software* utiliza, pode ser necessário que esse seja assinado por uma entidade veri-

ficadora para que possa ser executado. Isso tornaria o processo de desenvolvimento deste trabalho possivelmente mais complicado ao passo que, para isso, é necessário obter uma licença de desenvolvedor junto a essas entidades (TANENBAUM, 2010).

Outro ponto que deve ser levado em consideração é que o *Symbian* não possui implementação de memória virtual nem área de troca (*swap*) (TANENBAUM, 2010). Isso poderia tornar inviável o processamento de números grandes já que a quantidade de memória principal nesses dispositivos costuma ser limitada.

4.2 Escolha da Plataforma

A escolha do *Android* como plataforma para o desenvolvimento deste trabalho sustenta-se nos dados da pesquisa do instituto Gartner, que o apontam como destaque entre os sistemas operacionais que equipam os *smartphones* modernos. Segundo essa pesquisa, o SO desenvolvido pela *Google* apresentou um crescimento de 439% em suas vendas no segundo trimestre de 2011 em comparação com o mesmo período de 2010. Atualmente o *Android* equipa 43% dos *smartphones* vendidos contra 22% do segundo colocado, o *Symbian*, que apresentou uma queda em suas vendas de aproximadamente 6% neste mesmo período (GARTNER, 2011)

Assim, a escolha da plataforma *Android* se deve principalmente ao grande número de dispositivos equipados com este SO. Como já mencionado anteriormente, a quantidade de usuários representa um importante fator de sucesso para projetos de PRC. Outra vantagem no uso da plataforma *Android* é o total suporte à linguagem de programação JAVA, bem como a possibilidade da utilização do NDK, que é uma ferramenta complementar ao SDK Android, gratuita e com boa documentação, para desenvolvimento de aplicações nativas nas linguagens C e C++ (DEVELOPERS, 2011).

Também não pode deixar de ser observada a limitação do *iOS* quanto à multitarefa. Inicialmente, o *iOS* não possuía suporte a multitarefa, sendo adicionado posteriormente apenas para aplicativos nativos e através de um conjunto limitado de APIs (NASCIMENTO RIBEIRO, 2011).

Em relação ao *Symbian*, além do número de usuários estar em declínio, existem os já mencionados problemas de compatibilidade entre as versões do sistema operacional (DAMASIO, 2006). Além disso, a ausência de um mecanismo de memória virtual e de uma área de troca (*swap*) (TANENBAUM, 2010) podem inviabilizar a computação de números grandes.

A plataforma *Android* foi escolhida pelos fatos já mencionados, porém, não significa que outras plataformas não possam ser utilizadas para computação do projeto PRC criado neste trabalho. Para isso, basta que seja implementada uma versão do cliente BOINC para essa plataforma.

5 ESTUDO DE CASO

A criação de um projeto PRC com o *framework* BOINC presume, basicamente, as seguintes etapas:

a) instalação e configuração do servidor BOINC.

b) criação de um projeto BOINC no servidor.

c) definição das *workunits* (WU), validador e assimilador.

d) desenvolvimento da aplicação cliente.

As 3 primeiras etapas (etapas *a*, *b* e *c*) são referentes à criação e configuração da parte que compete ao servidor do projeto e serão descritas na Seção 5.1 deste capítulo. A última etapa (etapa *d*) refere-se à parte que é executada no cliente, e cuja implementação será detalhada na Seção 5.2. Ao fim, na Seção 5.3, serão descritos os testes de performance com a finalidade de comparar o desempenho obtido na plataforma *Android* com outras plataformas já utilizadas em projetos de PRC.

5.1 Servidor do Projeto

Nas Seções que seguem, serão detalhados os procedimentos realizados para a instalação e configuração do servidor do projeto.

5.1.1 Instalação e Configuração do Servidor

Inicialmente, foi necessário instalar e configurar o servidor BOINC. No *website* do projeto pode-se fazer o *download* de uma máquina virtual (VM) que contém:

a) a distribuição Debian 6 64-bit do Linux (modo caractere).

b) banco de dados *MySQL* (MYSQL, 2012).

c) um cliente do sistema de controle de versão *Subversion* (SVN) (SUBVERSION, 2012) pré-configurado, permitindo o *download* do código-fonte do servidor e do cliente BOINC, além de aplicações de demonstração.

d) *scripts* para compilar os códigos-fonte do servidor BOINC.

e) o conjunto de bibliotecas necessárias para compilar e executar o servidor BOINC.

A utilização dessa VM facilitou o processo de instalação e configuração do servidor uma vez que, salvo em algumas situações, que serão descritas neste capítulo, não foi preciso preocupar-se com detalhes relativos à instalação e configuração do sistema operacional e do gerenciador de banco de dados. Além disso, o cliente SVN instalado já vem pré-configurado com o repositório de códigos-fonte oficial do BOINC, possibilitando o *download* da última versão deste através da execução de um *script*.

5.1.2 Criação do Projeto PRC

O servidor BOINC contém aplicativos para a criação de projetos e WUs. Ao fazer o *download* do servidor BOINC, juntamente com os códigos-fonte do servidor são fornecidos *scripts* que facilitam a execução desses aplicativos. Além dos *scripts* fornecidos com o servidor, alguns outros foram desenvolvidos de modo a facilitar criação de WUs (processo esse que foi repetido exaustivamente durante o desenvolvimento do projeto), para executar consultas ao banco de dados do projeto ou mesmo para contornar problemas de permissão de usuário existentes na instalação do MySQL que acompanha a VM.

A criação de um novo projeto PRC no servidor BOINC é uma tarefa relativamente simples. Um *script* que acompanha o servidor encarrega-se da criação do banco de dados, da estrutura de diretórios e até mesmo do arquivo de configuração do *Apache HTTP Server* (FOUNDATION, 2012) para acesso via *Web*. As operações não executadas pelo *script* são documentadas de forma clara em um arquivo-texto com extensão *.readme* que fica localizado no diretório do projeto, devendo ser executadas manualmente.

Assim sendo, para a criação de um novo projeto é necessário executar o *script tools/make_project*, distribuído juntamente com os fontes do servidor. De acordo com as instruções encontradas no *website* do BOINC, basta passar por parâmetro o nome do projeto a ser criado, porém, ao realizar esse procedimento o *script* termina com erro devido a um problema de permissão de acesso ap MySQL. A solução para esse problema consiste em passar por parâmetro para esse *script* o usuário e a senha do MySQL, encontrados no arquivo /etc/mysql/debian.cnf além da URL base para acesso ao projeto pela *Web*. Para facilitar essa tarefa, foi criado o *script criar_projeto* que, recebendo por parâmetro apenas o nome do projeto, executa o *script make_project* passando os demais parâmetros necessários. O código-fonte do *script criar_projeto* pode ser consultado abaixo.

```
#!/usr/bin/env bash
cd /home/boincadm/boinc-trunk/tools
./make_project --url_base http://192.168.25.9
    --db_user debian-sys-maint
    --db_passwd F6u11nyvN5BGhEzH
    $1
```

Tendo criado com sucesso o projeto, o próximo passo consiste na inclusão de uma nova plataforma. Quando o cliente BOINC requisita uma tarefa ao servidor, ele envia informações referentes à sua plataforma para que o servidor identifique qual versão da aplicação deve ser enviada. A versão para *Android* do cliente BOINC identifica-se como *arm-android-linux-gnu*, dessa forma, foi necessário adicionar essa plataforma ao projeto através da inclusão das linhas abaixo no arquivo *project.xml*.

```
<platform>
    <name>arm-android-linux-gnu</name>
    <user_friendly_name>
        SO Android sendo executado sobre arquitetura ARM
    </user_friendly_name>
</platform>
```

5.1.3 Definição das Workunits

As *Workunits* (WU) representam a entrada de informação da aplicação. Elas são geradas e distribuídas pelo servidor e processadas pela aplicação no cliente, que recebe as WUs na forma de um ou mais arquivos-texto, compactados ou não, e gera o resultado do processamento dessa WU pela aplicação em um outro arquivo-texto, denominado resultado (SANTOS, 2005).

Para facilitar a criação das WUs e a interpretação dos resultados pelo servidor, tanto as WUs quanto os resultados tem sua forma definida em arquivos XML (*Extensible Markup Language*) denominados *templates*, sendo que devem ser definidos um *template* para a geração de WUs e outro para os resultados.

No *template* da WU, são definidos vários parâmetros para o processamento das tarefas. Dentre eles, os mais importantes e que foram utilizados neste trabalho são (WIKI, 2012) (SANTOS, 2005):

- *file_ref*: identifica os arquivos de entrada, definindo os seus nomes e a sua ordem.
- *rsc_memory_bound*: define a quantidade máxima de memória RAM (em *bytes*) que a aplicação poderá utilizar no dispositivo cliente.

- *rsc_disk_bound*: define o espaço máximo (em *bytes*) que a aplicação poderá ocupar na unidade de armazenamento (memória secundária) do dispositivo cliente.

- *min_quorum*: define o número mínimo de resultados iguais que o servidor esperará para considerar um resultado canônico.

- *target_nresults*: define o número de registros de resultados que serão criados no banco de dados do projeto no momento de criação da WU. Ao criar a WU, os resultados já são criados no banco de dados do projeto recebendo, inicialmente, o *status* "Pendente de envio". Esse número deve ser igual ou maior ao valor do parâmetro *min_quorum*. Caso o servidor atinja esse número de resultados sem ter encontrado um resultado canônico, o servidor deixará de enviar essa WU.

- *max_error_results*: define o número máximo de resultados finalizados com erro de uma WU. Ao atingir esse limite, o servidor deixará de enviar essa WU

- *max_success_results*: define o número máximo de resultados finalizados com sucesso de uma WU. Ao atingir esse limite, o servidor deixará de enviar essa WU.

- *max_total_results*: define o número máximo de resultados que essa WU pode receber, com erro ou não. Ao atingir esse limite, o servidor deixará de enviar essa WU.

- *delay_bound*: tempo máximo que o servidor esperará entre o envio de uma WU e sua resposta antes de enviá-la novamente a outro cliente.

O *template* utilizado para a criação das WUs deste projeto pode ser visualizado no quadro abaixo.

```
<file_info>
    <number>0</number>
</file_info>
<workunit>
    <file_ref>
        <file_number>0</file_number>
        <open_name>in</open_name>
    </file_ref>
    <rsc_memory_bound>30000000.000</rsc_memory_bound>
    <rsc_disk_bound>30000000.000</rsc_disk_bound>
    <target_nresults>1</target_nresults>
```

```
    <min_quorum>1</min_quorum>
    <max_error_results>3</max_error_results>
    <max_total_results>3</max_total_results>
    <max_success_results>3</max_success_results>
    <delay_bound>172800</delay_bound>
</workunit>
```

Além do *template* das WUs, é necessário criar um *template* para os resultados. Esse, mais simples, contém informações como os nomes lógicos e endereço para *upload* dos arquivos de saída (BOINC, 2012b).

- *name*: nome do arquivo de saída. O nome deve ser único por projeto. Ao utilizar o valor "<OUTFILE_0/>", o BOINC substituirá o nome do arquivo por um nome gerado com base na WU (BOINC, 2012b).
- *generated_locally*: indica que o arquivo de saída também será gerado localmente além de ser enviado para o servidor.
- *upload_when_present*: indica que será feito *upload* do arquivo de saída sempre que a aplicação for finalizada, mesmo que ela finalize com erro.
- *max_nbytes*: tamanho máximo do arquivo de saída (em *bytes*).
- *url*: *url* do diretório onde o arquivo de saída estará disponível no servidor.
- *open_name*: nome lógico do arquivo de saída que será acessado durante o processamento da WU. Deve ser único por resultado, podendo se repetir a cada nova WU, uma vez que cada uma das WUs será processada em um diretório próprio.

```
<file_info>
    <name><OUTFILE_0/></name>
    <generated_locally/>
    <upload_when_present/>
    <max_nbytes>10000</max_nbytes>
    <url><UPLOAD_URL/></url>
</file_info>
<result>
    <file_ref>
        <file_name><OUTFILE_0/></file_name>
        <open_name>out</open_name>
    </file_ref>
</result>
```

A API do BOINC provê uma função para criação de WUs (*create_work*()), que pode ser executada no momento do processamento de um resultado para a criação da próxima tarefa (BOINC, 2012b). Neste trabalho, porém, foi utilizado o *script create_work)*, fornecido com o servidor. Esse *script* permite criar todas as WUs antes do início do processamento do projeto.

Para tanto, é necessário que os *templates* de criação das WUs e dos resultados estejam no diretório *download* do projeto juntamente com os arquivos que contém os dados de entrada da WU a ser criada. Feito isso, deve-se passar por parâmetro para o *script* o nome dos *templates*, o nome do arquivo de entrada e o nome da WU, que deve ser único no projeto. Para facilitar essa tarefa, foi criado o *script criar_wu*, cujo código pode ser observado no quadro abaixo. Esse *script* exige como parâmetros de entrada apenas o nome da WU a ser criada (que deve ser único por projeto) e o nome do arquivo que contém os dados de entrada dessa WU. Sua finalidade é facilitar a criação das WUs, processo esse que foi repetido exaustivamente durante o desenvolvimento deste trabalho.

```
#!/usr/bin/env bash
./bin/create_work --appname pepin_app --wu_name $1
    --wu_template templates/pepin_wu
    --result_template templates/pepin_result
    $2
```

Por fim, é feita a configuração dos *daemons* que serão executados continuamente no servidor. O escalonador, o transacionador e o *file deleter* são configurados automaticamente pelo *script* de criação do projeto. A inicialização do validador e do assimilador deve ser configurada manualmente. Inicialmente, o projeto foi configurado para utilizar o validador e o assimilador de exemplo fornecidos com o servidor BOINC. Para tanto, foi necessário adicionar as linhas abaixo ao arquivo *project.xml*.

```
    <daemon>
      <cmd>
        sample_trivial_validator -d 3 --app pepin_app
      </cmd>
    </daemon>
    <daemon>
      <cmd>
        sample_assimilator -d 3 --app pepin_app
      </cmd>
    </daemon>
```

Para facilitar o acompanhamento da execução das WUs, neste projeto optou-se por desativar o *file deleter*, comentando as linhas abaixo no arquivo *project.xml*.

```
<!-- <daemon>
  <cmd>
    file_deleter -d 3
  </cmd>
</daemon> -->
```

5.1.4 Validador e Assimilador

Como descrito na Seção 3.2.1, cada projeto BOINC possui uma implementação específica que pode ser dividida entre a parte que é executada no cliente e a parte de *back end* do projeto, que é executado no servidor. Do lado do cliente, tem-se o programa responsável por realizar o processamento das WUs, cuja implementação será descrita na Seção 5.2.1 deste capítulo. Já no lado do servidor, tem-se o validador e o assimilador, cujas funções são detalhadas na Seção 3.2.1.

Por tratar-se de um projeto acadêmico em ambiente controlado, onde não há risco fraudes, não será necessário implementar um validador. Ao invés disso, será utilizado o validador de exemplo que acompanha o servidor BOINC. Esse validador apenas lê o banco de dados periodicamente em busca de novos resultados, modificando, invariavelmente, o estado de suas WUs para "deve ser assimilado" (*ready for assimilation*) (SANTOS, 2005).

Por motivos que serão descritos na Seção 5.2.1, o processamento das WUs desse projeto não pôde ser paralelizado. Assim sendo, não será necessário desenvolver um assimilador, uma vez que cada WU compõe um resultado completo de um número de Fermat. Dado esse cenário, é possível utilizar o assimilador de exemplo fornecido com o servidor BOINC. Esse assimilador não executa ação alguma sobre os resultados, apenas marca a WU como "assimilada" assim que o seu primeiro resultado for recebido.

Embora os projetos de computação pública tenham uma maior utilidade prática quando utilizam computação paralela de forma distribuída, a dificuldade encontrada em paralelizar o cálculo proposto em nada prejudica o principal objetivo do trabalho, que consiste em criar um projeto de PRC cujos clientes sejam dispositivos móveis baseados em *Android*.

5.1.5 Registro da Aplicação no Servidor BOINC

O registro da aplicação no servidor consiste em criar os registros necessários no banco de dados do projeto para que o servidor consiga gerenciar a distribuição das versões corretas dos executáveis da aplicação para os clientes de acordo com a sua plataforma. Esse registro é feito através da execução de *scripts*, de forma que, antes de executá-los, é necessário declarar cada uma das aplicações em um arquivo XML,

que será a base de execução desse *script*.

Para registrar a aplicação no servidor, é necessário executar as seguintes etapas:

1) declarar e nomear a aplicação incluindo uma entrada no arquivo *project.xml*.

2) compilar a aplicação para cada uma das plataformas para as quais ela será disponibilizada.

3) copiar os binários da aplicação para o diretório *app* do projeto, respeitando uma estrutura de sub-diretórios pré-definida.

4) executar os *scripts* responsáveis por registrar a aplicação no servidor e controlar o seu versionamento.

A declaração da aplicação foi feita pela inclusão das linhas abaixo no arquivo *project.xml*, seguida da execução do *script bin/xadd*, que é responsável por fazer a leitura desse arquivo e registrar as aplicações declaradas no banco de dados do projeto.

```
<app>
    <name>pepin_app</name>
    <user_friendly_name>
        Verifica a primalidade de numeros de Fermat
    </user_friendly_name>
</app>
```

A compilação da aplicação em cada uma das plataformas utilizadas é descrita na Seção 5.2.2 deste capítulo. Essa compilação deve ser feita antes do registro da aplicação, porém, será detalhada na próxima Seção para que haja uma divisão clara entre a configuração do servidor e o desenvolvimento da parte que compete ao cliente.

Tendo finalizado a compilação, é necessário copiar os binários da aplicação, inclusive as bibliotecas utilizadas, para o diretório *app* do projeto e executar o *script bin/update_version*, que é responsável por fazer o controle de versão da aplicação que será distribuída aos clientes. Após a execução do *script*, os executáveis atualizados da aplicação ficam disponíveis para o cliente na pasta *download* do projeto.

Como a plataforma *Android* não é suportada nativamente pelo BOINC, foi necessário criar um arquivo descritor da aplicação, nomeado *app_info.xml*. Esse descritor relaciona todos os arquivos da aplicação, identificando os executáveis e o programa principal, que será executado pelo cliente BOINC no dispositivo móvel. Esse arquivo deve ser criado no mesmo diretório que o executável e será enviado para o cliente juntamente com todos os outros arquivos que compõe a aplicação. No Anexo C pode-se observar o conteúdo do arquivo utilizado na aplicação desenvolvida.

Como nas versões para *Windows* e *Linux* foram utilizadas apenas bibliotecas estáticas, nenhum arquivo adicional foi criado além do executável, sendo necessário apenas copiar o seu executável para a pasta *app* e executar o *script bin/update_version* para registrá-la no servidor. Por tratar-se de uma plataforma reconhecida nativamente pelo servidor BOINC, nem mesmo a criação do arquivo *app_info.xml* foi necessária.

5.2 Aplicação Cliente

A aplicação cliente é a parte responsável pelo processamento das WUs. Essa aplicação foi desenvolvida em linguagem de programação C++ utilizando a API BOINC e a biblioteca GMP. O código-fonte completo da aplicação pode ser consultado no Anexo A e os detalhes da sua implementação e compilação serão descritos nesta Seção.

5.2.1 Desenvolvimento da Aplicação

O funcionamento básico da aplicação consiste em ler de um arquivo-texto o expoente do número de Fermat sobre o qual será aplicado o teste de Pepin, realizar o teste e gerar um arquivo de saída com o resultado do processamento, que conterá o valor "P" quando o número de Fermat for primo e "C" quando esse número for composto.

O algoritmo utilizado para a execução do teste de Pepin foi aquele já detalhado na Seção 2.5.1, com algumas modificações para que a API BOINC fosse utilizada na leitura dos dados de entrada e na escrita dos resultados da WU, para realização de *checkpoints* e para atualização do progresso do processamento no cliente. Para verificar a congruência definida no teste de Pepin o algoritmo necessita, a cada iteração, do resultado obtido na iteração anterior. Assim sendo, não foi encontrada uma forma de paralelizar o processamento de teste de Pepin para um número de Fermat. Ao invés disso, a abordagem adotada foi a de que cada WU conterá um expoente e esse expoente resultará em um número de Fermat cuja primalidade será integralmente verificada pelo cliente que recebeu a WU.

Conforme detalhado na Seção 5.1.3, uma WU será composta por um arquivo de entrada, que conterá o expoente do número de Fermat cuja primalidade se deseja verificar, e por um arquivo de saída, que conterá o resultado do processamento, que poderá ser "P" se o número de Fermat for primo ou "C" se o número for composto. Para ler o arquivo de entrada da WU, utilizou-se a função *boinc_resolve_filename*, conforme detalhado no Algoritmo 5.2.1.

Algoritmo 5.2.1 Aplicação Cliente - Leitura de Arquivo

```
boinc_resolve_filename(INPUT_FILENAME,input_path, sizeof(input_path));
infile = boinc_fopen(input_path, "r");
```

Para escrever os resultados no arquivo de saída, utilizou-se a estrutura *MFILE*, que permite manter em memória todo o conteúdo do arquivo, gerando o arquivo físico a partir de uma função *flush*, o que garante a atomicidade da operação, evitando que sejam gerados arquivos de saída incompletos. A implementação dessa estrutura é detalhado no Algoritmo 5.2.2.

Algoritmo 5.2.2 Aplicação Cliente - Escrita em Arquivo

```
MFILE out;
...
 boinc_resolve_filename(OUTPUT_FILENAME,output_path,
                        sizeof(output_path));
 retval = out.open(output_path, "wb");
...
    out._putchar('P');
...
    out._putchar('C');
...
 retval = out.flush();
```

Além disso, a função *boinc_time_to_checkpoint* é invocada a cada iteração para verificar se é o momento de realizar um *checkpoint*. Por padrão, o BOINC gerencia o tempo entre os *checkpoints*.

Algoritmo 5.2.3 Aplicação Cliente - Uso da API BOINC para *Checkpoints*

```
      if (boinc_time_to_checkpoint()) {
        retval = do_checkpoint(&i, &x);
...
      boinc_checkpoint_completed();
```

Caso o resultado da função *boinc_time_to_checkpoint* retorne verdadeiro, então a função *do_checkpoint*, cuja implementação pode ser consultada no Anexo A, é invocada, gravando em um arquivo-texto o estado de memória das variáveis necessárias para que o processamento seja retomado no caso de uma interrupção. Não ocorrendo erros nesse processo, a função *boinc_checkpoint_completed* comunica ao cliente BOINC que o *checkpoint* foi realizado com sucesso.

A função *boinc_fraction_done* é invocada também a cada iteração, com o objetivo de atualizar o percentual concluído do processamento, permitindo o acompanhamento do progresso da WU no cliente.

Algoritmo 5.2.4 Aplicação Cliente - Acompanhamento da Execução

```
fractionDone = (mpz_get_d(i) / (double) n);
boinc_fraction_done(fractionDone);
```

A cada iteração, atribui-se à variável *fractionDone* o percentual de conclusão do processamento. Em seguida, essa variável é passada como argumento para a função *boinc_fraction_done*, que atualiza o percentual concluído do processamento, exibindo assim o progresso da WU no cliente BOINC.

Para a manipulação de números grandes na versão para *Android*, foi utilizada uma versão da biblioteca GMP pré-compilada para o SO *Android* (MOHR, 2012). Para as demais versões da aplicação, foi possível utilizar as versões da GMP disponibilizadas no *website* do projeto (GMP, 2011).

5.2.2 Compilação da Aplicação

A compilação foi realizada através do *Android* NDK, que, além do compilador para a arquitetura ARM, provê uma ferramenta para a compilação que permite relacionar todas as dependências, *includes* e bibliotecas em um arquivo-texto, facilitando o processo de *build*. Essa ferramenta desempenha uma função similar ao GNU *Make* (MAKE, 2012) , largamente utilizado para facilitar a compilação de *softwares* de código aberto. Também acompanham o NDK, as bibliotecas padrão da linguagem C++, pré-compiladas para a arquitetura ARM. Essas bibliotecas foram utilizadas no desenvolvimento da aplicação.

Como os binários do *Android* NDK são fornecidos apenas para plataformas 32-bit, utilizou-se a distribuição *Linux Ubuntu 10.10 32-bit* para o desenvolvimento da aplicação. A IDE (*Integrated Development Environment*) adotada no desenvolvimento foi o *Eclipse* e o NDK foi utilizado para compilar o programa e gerar os executáveis para a arquitetura ARM. Também foi desenvolvido um *script* para compilar a aplicação e outro para enviá-la ao servidor BOINC, num processo que pode ser considerado um *deploy* da aplicação e pode ser executado através do próprio *Eclipse*, formando assim um ambiente de desenvolvimento integrado.

Conforme recomendado no manual do NDK, foi criado um arquivo *Android.mk* onde foram descritas todas as regras de compilação da aplicação. Nesse mesmo arquivo foram referenciadas as bibliotecas e *includes* necessárias. Além disso, para facilitar a utilização do *script* de compilação fornecido com o NDK, o projeto seguiu a estrutura de diretórios recomendada no manual do NDK, de tal forma que os códigos-fonte da aplicação e o arquivo *Android.mk* foram armazenados em um diretório chamado *jni*. O conteúdo desse arquivo pode ser consultado no anexo B.

Uma vez definidas as regras de compilação, a execução do *script ndk-build*, fornecido com o NDK, gera o binário executável para a arquitetura ARM. Esse binário é

gerado no diretório *libs* do projeto, onde também encontram-se todas as bibliotecas utilizadas na aplicação.

Também foram compiladas versões da aplicação para os SOs *Windows* e *Linux*, ambos em plataforma *x86 32-bit*. Essas versões foram utilizadas para realizar testes comparativos de performance entre a aplicação executada em um celular e a mesma aplicação executada em um computador do tipo PC. Para tanto, foi necessário utilizar versões correspondentes de todas as bibliotecas utilizadas pela aplicação para cada um dos SOs. As versões para *Windows* e para *Linux* da API BOINC são distribuídas com o servidor BOINC. As versões pré-compiladas da GMP para ambos os SOs estão disponíveis para *download* no *website* oficial da GMP. Para realizar a compilação da aplicação para *Windows* foi utilizado o *Microsoft Visual Studio 2010 Express*, que é o ambiente de desenvolvimento recomendado pelo BOINC e para o qual já existe um projeto de exemplo disponibilizado com o servidor. Para o *Linux*, foi criado um novo projeto no *Eclipse*, sendo necessário apenas incluir no projeto as versões correspondentes das bibliotecas utilizadas.

5.3 Resultados e Tempos de Execução

Tendo atingido o objetivo principal deste trabalho, que é o desenvolvimento de um projeto de PRC para a verificação de primalidade de números de Fermat utilizando como clientes dispositivos móveis baseados em *Android*, e a fim de verificar a viabilidade prática de se utilizar tais dispositivos para projetos de PRC, foram medidos os tempos de execução da aplicação desenvolvida em três diferentes plataformas.

Os testes comparativos foram executados utilizando um celular modelo *Sony Ericsson Xperia X10*, que conta com um processador *Snapdragon* com núcleo único de 1GHz, 256MB de memória RAM e SO *Android* na versão 2.3.5 e em um *notebook* equipado com um processador *Intel Core 2 Duo* que possui dois núcleos com *clock* de 1,3GHz cada, SO *Windows* 7 e 4GB de memória RAM.

Os testes com o SO *Linux* foram executadas em uma VM neste mesmo *notebook*, que foi configurada para utilizar 2GB de memória RAM e até 100% do processamento disponível. A distribuição do *Linux* utilizada foi a *Ubuntu*, versão 10.10 com *kernel* na versão 2.6.35-22 e plataforma x86 32-bit. Em todos os testes a aplicação foi executada através do cliente BOINC, sendo que este foi configurado de forma a permitir a utilização de todos os recursos computacionais disponíveis. Assim garantiu-se que o processamento não tenha sido limitado pelo cliente BOINC.

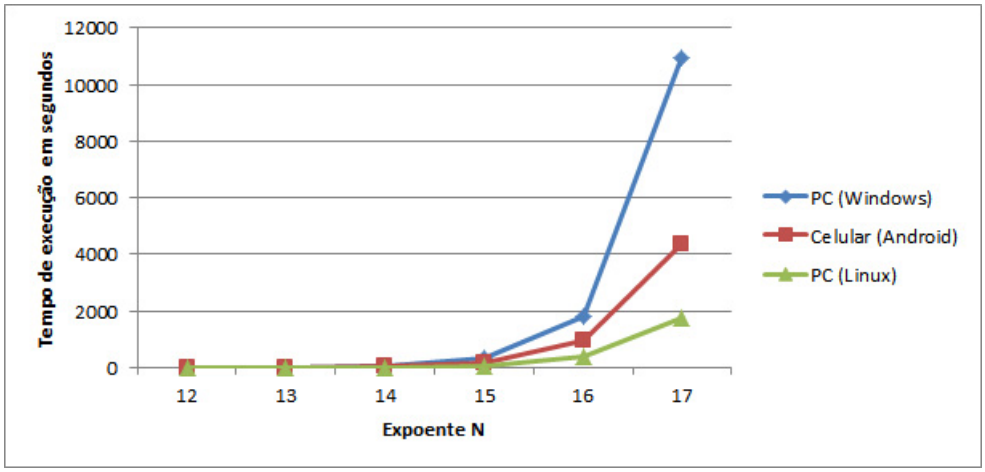

Figura 5.1: Tempos de Execução da Aplicação Através do Cliente BOINC

Pode ser observado na figura 5.1 que, na média das medições realizadas, o *smartphone* apresentou um desempenho 190% superior ao *notebook* quando este executava SO *Windows* e 241% inferior aos resultados obtidos neste mesmo *notebook* executando SO *Linux*.

Foram levantadas duas possibilidades para o baixo desempenho da aplicação no *Windows*. A utilização da API do BOINC ou a falta de otimização do código gerado pelo compilador do *Visual C++*. Foram realizados testes adicionais executando a aplicação diretamente pelo *prompt* de comando (cmd) do *Windows*, sem qualquer relação com o cliente BOINC. Mesmo nesse último caso a aplicação apresentou tempos praticamente idênticos ao obtidos com a execução através do cliente BOINC, indicando que o problema de performance pode não ser causado pelo cliente BOINC, mas sim, pelo compilador do *Visual C++*. Problemas de performance semelhantes ao utilizar o compilador do *Visual C++* também foram relatados por Jonathan Titton em sua monografia, intitulada *Processamento de Eletroencefalograma utilizando o método Matching Pursuit em uma plataforma Grid* (TITTON, 2006).

No gráfico da figura 5.2, que precisou ser colocado em perspectiva devido aos resultados muito semelhantes entre as duas execuções, pode-se observar que praticamente não há diferença entre os tempos da aplicação executada através do cliente BOINC ou diretamente pelo *prompt* de comando do *Windows*. Testes semelhantes foram realizados no *Linux*, onde também não foi percebida queda de performance em função da utilização da API BOINC.

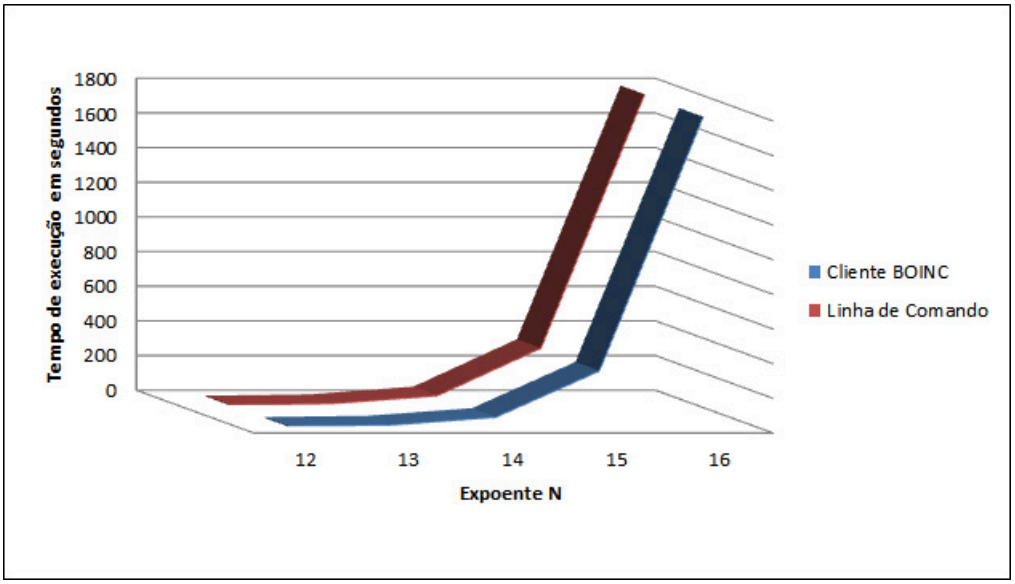

Figura 5.2: Tempos de Execução da Aplicação Através do *Prompt* de Comando

6 CONCLUSÃO

O poder de processamento dos dispositivos móveis tem crescido de forma acelerada nos últimos anos. Utilizar esses dispositivos em projetos de PRC consiste em uma opção factível e, embora os computadores tradicionais ainda possuam um poder de processamento maior em termos gerais, comprovou-se que, para fins específicos e sob determinadas condições, a performance desses dispositivos pode ser equivalente ou até mesmo superior à de um PC moderno. Além disso, o grande número de dispositivos móveis em uso e a observada tendência de crescimento do segmento os tornam bons candidatos a clientes em projetos desse tipo.

A aplicação escolhida para demonstrar a possibilidade do uso de computação pública em dispositivos móveis baseados em *Android* foi a busca por números primos de Fermat. Como já foi demonstrado, números dessa forma crescem muito rapidamente, atingindo facilmente a ordem de milhares de algarismos. A importância da busca de números primos cada vez maiores justifica-se no seu uso para criptografia, conforme foi citado na Seção 2.2 onde é descrito o funcionamento do método RSA.

O tamanho dos números envolvidos no problema exige a utilização de uma biblioteca especial para manipular números de grande magnitude. Dentre as alternativas disponíveis, foi optado pela biblioteca GMP, uma vez que grande parte do trabalho foi desenvolvido utilizando a linguagem de programação C, para a qual essa biblioteca oferece suporte.

Para a implementação do paradigma de computação pública, foi escolhida a plataforma BOINC. A escolha foi motivada pelo fato do BOINC reunir todos os componentes necessários à computação pública em um *framework* de fácil acesso, aberto e bem documentado e que é amplamente utilizado por outros projetos.

Não foi possível paralelizar a implementação do teste de Pepin, que foi utilizado neste trabalho para verificar a primalidade dos números de Fermat. Porém, isso não inviabiliza o principal objetivo do trabalho, que consistiu na implementação de um sistema PRC cujos clientes sejam dispositivos móveis baseados em *Android*. O projeto PRC foi implementado com sucesso e as medições de performance provaram a viabilidade de se utilizar tais dispositivos como clientes para projetos PRC.

Embora a aplicação escolhida para este trabalho não se beneficie com o aumento do número de participantes, uma vez que não foi possível paralelizá-la, futuros trabalhos podem ser desenvolvidos de forma a explorar o uso desses dispositivos de forma paralela, em aplicações que se beneficiem do número de clientes conectados ao projeto.

REFERÊNCIAS

AGRAWAL, M.; KAYAL, N.; SAXENA, N. Primes in p. In: MATHEMATICS, 2002, Kanpur - India. **Annals...** [S.l.: s.n.], 2002.

ANDERSON, D.; FEDAK, G. The computational and storage potential of volunteer computing. In: IEEE/ACM INTERNATIONAL SYMPOSIUM ON CLUSTER COMPUTING AND THE GRID, 2006, Singapore. **Anais...** [S.l.: s.n.], 2006.

ANDERSON, D. P. Boinc: a system for public-resource computing and storage. In: IEEE/ACM INTERNATIONAL WORKSHOP ON GRID COMPUTING, 5., 2004, Pittsburgh - USA. **Anais...** [S.l.: s.n.], 2004. p.1–7.

AXIOM. **Axiom, the scientific computation system**. Disponível em: http://axiom-developer.org/. Acesso em 22 nov. 2011, website.

BOINC. **Boinc on android**. Disponível em: http://boinc.berkeley.edu/trac/wiki/AndroidBoinc. Acesso em 07 out. 2012, website.

BOINC. **Boinc softwaredevelopment wiki**. Disponível em: http://boinc.berkeley.edu/trac/wiki/SoftwareDevelopment. Acesso em 04 nov. 2012, website.

BOINCOID. **Boincoid - an android port of the boinc platform**. Disponível em: http://boincoid.sourceforge.net/. Acesso em 03 nov. 2012, website.

BUYYA, R. **High performance cluster computing**: architectures and systems. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

CALDWELL, C. **The prime pages**: prime number research, records, and resource. Disponpivel em: http://primes.utm.edu. Acesso em: 02 out. 2011., website.

CARDOSO, C. **Fatoração de números inteiros usando curvas elíticas**. 2003. Dissertação (Mestrado em Ciência da Computação) — Departamento de Computação e Estatística, Universidade Federal de Mato Grosso do Sul, Campo Grande - MS - Brasil.

COOLIDGE, J. L. **The mathematics of great amateurs**. [S.l.]: Dover, New York :, 1963. viii,211 p. :p.

COUTINHO, S. C. **Primalidade em tempo polinomial**: uma introdução ao algoritmo aks. Rio de Janeiro - RJ - Brasil: Sociedade Brasileira de Matemática, 2004. (Coleção Iniciação Científica).

COUTINHO, S. C. **Números inteiros e criptografia rsa**. 2a.ed. Rio de Janeiro - RJ - Brasil: Instituto Nacional de Matemática Pura e Aplicada, 2005. (Série de Comnputação e Matemática).

CUCCARO, V. D. **Developing google android mobile clients for web services**: a case study. Nápoles - Itália: [s.n.], 2008. Monografia - Graduação - Curso de Engenharia da Computação, Universita Degli Studi di Napoli Dederico II.

DAMASIO, F. **Programação mms com o symbian os**. Porto Alegre - RS - Brasil: [s.n.], 2006. Notas de Aula - Instituto de Informática - Universidade Federal do Rio Grande do Sul (UFRGS).

DEVELOPERS, A. **Página oficial do android developers**. Disponível em: http://developer.android.com/. Acesso em 02 nov. 2011, website.

DISTRIBUTED. **Página oficial do distributed.net**. Disponível em: http://distributed.net/. Acesso em 31 out. 2011, website.

ECLIPSE ORG. **The eclipse foundation**. Disponível em: http://www.eclipse.org/. Acesso em 29 nov. 2011, website.

FOLDING@HOME. **Folding@home**: página oficial do projeto. Disponível em: http://folding.stanford.edu/. Acesso em 01 set. 2011, website.

FOLDIT. **Foldit**: solve puzzles for science. Disponível em: http://fold.it/portal/. Acesso em 13 nov. 2011, website.

FOSTER, I.; KESSELMAN, C. (Ed.). **The grid**: blueprint for a new computing infrastructure. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1999.

FOUNDATION, A. **Apache http server project**. Disponível em: http://httpd.apache.org/. Acesso em 18 nov. 2012, website.

GARTNER. Gartner says sales of mobile devices in second quarter of 2011 grew 16.5 percent year-on-year; smartphone sales grew 74 percent. **Gartner**, 2011. Disponível em: http://www.gartner.com/it/page.jsp?id=1764714. Acesso em 27 nov. 2011.

GARTNER. Gartner says worldwide smartphone sales soared in fourth quarter of 2011 with 47 percent growth. **Gartner**, 2012. Disponível em: http://www.gartner.com/it/page.jsp?id=1924314. Acesso em 03 nov. 2012.

GIMPS. **Great internet mersenne prime search**. Disponível em: http://www.mersenne.org/. Acesso em 22 out. 2011, website.

GMP. **The gnu multiple precision arithmetic library**. Disponível em: http://gmplib.org/. Acesso em 16 nov. 2011, website.

GREENE COMPUTING. **Linpack for android top 10**. Disponível em: http://www.greenecomputing.com/apps/linpack/linpack-top-10/. Acesso em 22 ago. 2011, website.

GREENE COMPUTING. **Linpack for android**. Disponível em: http://www.greenecomputing.com/apps/linpack/. Acesso em 29 nov. 2011, website.

GROENWALD, C. L. O.; OLIVEIRA SAUER, L. de; FRANKE, R. F. **A história da matemática como recurso didático para o ensino da teoria dos números e a aprendizagem da matemática no ensino básico**. Artigo.

HORTON, I. **Beginning c**: from novice to professional. 4a.ed. Berkeley - Califórnia - EUA: Apress, 2006.

IDG NOW. **Nokia finalmente anuncia "morte"do symbian nos eua**. Disponível em: http://idgnow.uol.com.br/computacao_pessoal/2011/08/10/nokia-finalmente-anuncia-morte-do-symbian-nos-eua/. Acesso em 28 nov. 2011, website.

KORPELA, E. J.; ANDERSON, D. P.; BANKAY, R.; COBB, J.; HOWARD, A.; LEBOFSKY, M.; SIEMION, A. P.; KORFF, J. von; WERTHIMER, D. **Status of the uc-berkeley seti efforts**. Artigo.

LECHETA, R. R. **Google android**: aprenda a criar aplicações para dispositivos móveis com o android SDK. 2a.ed. São Paulo - SP - Brasil: Novatec, 2011.

LONGBOTTOM, R. **Linpack benchmark results on pcs**. Disponível em: http://www.roylongbottom.org.uk/linpackresults.htm. Acesso em 22 ago. 2011, website.

MAKE, G. **Gnu make - gnu priject - free software foundation**. Disponível em: http://www.gnu.org/software/make/. Acesso em 18 nov. 2012, website.

MAPLESOFT. **Página institucional da maplesoft**. Disponível em: http://www.maplesoft.com/. Acesso em 22 nov. 2011, website.

MATEUS, G. R.; LOUREIRO, A. A. F. **Introdução a computação móvel**. Disponível em: http://homepages.dcc.ufmg.br/ loureiro/cm/docs/cm_livro_2e.pdf. Acesso em 28 nov. 2011., Departamento de Ciência da Computação, Universidade Federal de Minas Gerais.

MOBILTEC. **Mobiltec - mobilidade em negócios**. Disponível em: http://www.mobiltec.com.br/blog/index.php/desenvolvimento-mobile-nas-plataformas-android-e-ios/. Acesso em 27 nov. 2011, Blog.

MOHR, M. **A prebuilt gmp module for android**. Disponível em: https://github.com/Rupan/gmp. Acesso em 14 nov. 2012, website.

MYSQL. **Mysql, the world's most popular open source database**. Disponível em: http://www.mysql.com/. Acesso em 18 nov. 2012, website.

NASCIMENTO RIBEIRO, F. I. do. **Uma abordagem comparativa do desenvolvimento de aplicações para dispositivos móveis**. PE - Brasil: [s.n.], 2011. Monografia - Bacharelado - Curso de Ciência da Computação, Universidade Federal de Pernambuco.

NATIVE, B. **Native port of the boinc client for android devices and not only**. Disponível em: http://nativeboinc.org. Acesso em 07 out. 2012, website.

NOV, O.; ANDERSON, D.; ARAZY, O. **Volunteer computing**: a model of the factors determining contribution to community-based scientific research. Artigo.

PIZZAMIGLIO, F. **Uma solução paralela para a busca de números primos de mersenne**. Caxias do Sul - RS - Brasil: [s.n.], 2005. Monografia - Bacharelado - Curso de Ciência da Computação, Universidade de Caxias do Sul.

POMERANCE, C. Very short primality proofs. **Mathematics of Computation**, v.48, n.177, p.315–322, Janeiro 1987.

PREDICTION, C. **Climate prediction**: página oficial do projeto. Disponível em: http://climateprediction.net/. Acesso em 01 set. 2011, website.

QUICKOFFICE. **Quickoffice**: página oficial. Disponível em: http://www.quickoffice.com/. Acesso em 15 nov. 2011, website.

RIBENBOIM, P. **Números primos**: mistérios e recordes. Rio de Janeiro - RJ - Brasil: Associação Instituto Nacional de Matemática Pura e Aplicada, 2001.

ROCHA DIAS, R. A. da. **Algumas evidências computacionais da infinitude dos números primos de Fibonacci e generalizações destes**. Natal - RN - Brasil: [s.n.], 2008. Monografia - Bacharelado - Curso de Ciência da Computação,

Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte.

RSA. **Rsa, the security division of emc**. Disponível em: http://www.rsa.com/. Acesso em 22 nov. 2011, website.

SANTOS, A. A. da Luz dos. **Reconhecimento de padrões em dna utilizando computação pública**. Caxias do Sul - RS - Brasil: [s.n.], 2005. Monografia - Bacharelado - Curso de Ciência da Computação, Universidade de Caxias do Sul.

SENA, H. T. A. de. **Algumas evidências computacionais da infinitude dos números primos palindrômicos e generalizações destes**. Natal - RN - Brasil: [s.n.], 2008. Monografia - Bacharelado - Curso de Ciência da Computação, Departamento de Informática e Matemática Aplicada, Universidade Federal do Rio Grande do Norte.

SETI@HOME. **Seti@home**: página oficial do projeto. Disponível em: http://setiathome.berkeley.edu/. Acesso em 01 set. 2011, website.

SINGH, S. **O último teorema de fermat**: a história do enigma que confundiu as maiores mentes do mundo durante 358 anos. 18a.ed. Rio de Janeiro - RJ - Brasil: Editora Record, 1997.

SOTTRUP, C. U.; PEDERSEN, J. G. Developing Distributed Computing Solutions Combining Grid Computing and Public Computing. , v.1, n.1, p.1–6, 2005.

SUBVERSION, A. **Apache subversion**. Disponível em: http://subversion.apache.org/. Acesso em 18 nov. 2012, website.

TABORDA, S. **Sergio taborda's home page**: trabalhando com números. Disponível em: http://sergiotaborda.wordpress.com/desenvolvimento-de-software/java/faq/numeros/. Acesso em 22 out. 2011, website.

TANENBAUM, A. S. **Sistemas operacionais modernos**. 3a.ed. [S.l.]: Editora Pearson, 2010.

TAYLOR, N. J. **Public grid computing participation**: an exploratory study of determinants. Artigo.

TECNOBLOG. **Gamers solucionam problema de 10 anos...em 3 semanas**. Disponível em: http://tecnoblog.net/77298/gamers-foldit-problema/. Acesso em 22 out. 2011, website.

TITTON, J. **Processamento de eletroencefalograma utilizando o método matching pursuit em uma plataforma grid**. Caxias do Sul - RS - Brasil: [s.n.], 2006. Monografia - Bacharelado - Curso de Ciência da Computação, Universidade de Caxias do Sul.

TREVISTI, G. **Digital & dreams**: swiss army knife. Disponível em: http://www.digital-and-dreams.net/main/. Acesso em 15 nov. 2011, website.

VALENTE, L. **Notas sobre desenvolvimento em symbian os**. Rio de Janeiro - RJ - Brasil: [s.n.], 2007. Notas de Aula - Pontífica Universidade Católica.

VASIGA, T. M. J. **Error detection in number-theoretic and algebraic algorithms**. 2008. Doutorado — University of Waterloo, Waterloo - Ontario - Canada.

WIKI, U. B. **Unofficial boinc wiki**. Disponível em: http://www.boinc-wiki.info. Acesso em 04 nov. 2012, website.

WOLFRAM. **Wolfram mathworld**: fermat number. Disponível em: http://mathworld.wolfram.com/FermatNumber.html. Acesso em 22 out. 2011, website.

WORLD COMMUNITY GRID. **World community grid**: página oficial do projeto. Disponível em: http://www.worldcommunitygrid.org. Acesso em 01 set. 2011, website.

ANEXO A CÓDIGO-FONTE DA APLICAÇÃO

Código-fonte da Aplicação

```
ï»¿#include "config.h"
#include <cstdio>
#include <cctype>
#include <ctime>
#include <cstdlib>
#include <csignal>
#include <unistd.h>
#include <sstream>

#include "str_util.h"
#include "util.h"
#include "filesys.h"
#include "boinc_api.h"
#include "mfile.h"
#include "gmp.h"

using std::string;

#define CHECKPOINT_FILE "pepin_state"
#define INPUT_FILENAME "in"
#define OUTPUT_FILENAME "out"

MFILE out;
char buf[256], chkpt_path[512];
FILE* state, *infile;

int do_checkpoint(mpz_t *i, mpz_t *x) {
  int retval;
```

```
  string resolved_name;

  FILE* f = fopen("temp", "w");
  if (!f)
    return 1;
  fprintf(f, "%d \n", (sizeof(char)
      * mpz_sizeinbase(*i, 10)));
  fprintf(f, "%s \n", mpz_get_str(NULL, 10, *i));
  fprintf(f, "%d \n", (sizeof(char)
      * mpz_sizeinbase(*x, 10)));
  fprintf(f, "%s \n", mpz_get_str(NULL, 10, *x));
  fclose(f);

  fprintf(stderr,
      "%s APP: Checkpoint feito. i = %s \n",
      boinc_msg_prefix(buf, sizeof(buf)),
      mpz_get_str(NULL, 10, *i));

  boinc_resolve_filename_s(CHECKPOINT_FILE,
      resolved_name);
  retval = boinc_rename("temp",
      resolved_name.c_str());
  if (retval) {
    return retval;
  }

  return 0;
}

int load_checkpoint(mpz_t *i, mpz_t *x) {
  char *readI;
  char *readX;
  int tamanho;

  fprintf(stderr, "Entrou no load checkpoint \n",
      boinc_msg_prefix(buf, sizeof(buf)));

  boinc_resolve_filename(CHECKPOINT_FILE,
      chkpt_path, sizeof(chkpt_path));
```

```
  state = boinc_fopen(chkpt_path, "r");
  if (state) {
    fprintf(stderr,
        "Encontrou arquivo de checkpoint \n",
        boinc_msg_prefix(buf, sizeof(buf)));

    fscanf(state, "%d \n", &tamanho);
    readI = (char*) malloc(sizeof(char)
        * (tamanho + 1));
    fscanf(state, "%s \n", &readI);

    fscanf(state, "%d \n", &tamanho);
    readX = (char*) malloc(sizeof(char)
        * (tamanho + 1));
    fscanf(state, "%s \n", &readX);
    fclose(state);

    fprintf(stderr, "Leu do checkpoint \n",
        boinc_msg_prefix(buf, sizeof(buf)));
    mpz_set_str(*i, readI, 10);
    mpz_set_str(*x, readX, 10);

    fprintf(
        stderr,
        "Carregou do checkpoint para as variaveis \n",
        boinc_msg_prefix(buf, sizeof(buf)));

    fprintf(
        stderr,
        "%s APP: Load de checkpoint (dentro da funcao). "
        "Valores atribuidos i = %d \n",
        boinc_msg_prefix(buf, sizeof(buf)),
        mpz_get_str(NULL, 10, *i));

    return 1;
  }
  return 0;

}
```

```
int pepin(double k, mpz_t fn) {

  mpz_t i, x, fk;
  mpz_init(i);
  mpz_init(x);
  mpz_init(fk);
  mpz_set_ui(x, 5);
  long int n;
  int retval;
  double fractionDone = 0;
  n = pow(2, k) - 1;

  boinc_fraction_done(fractionDone);

  if (load_checkpoint(&i, &x)) {

    fprintf(
        stderr,
        "%s APP: Load de checkpoint (fora da funcao). "
        "i = %s, \n x = %s \n",
        boinc_msg_prefix(buf, sizeof(buf)),
        mpz_get_str(NULL, 10, i), mpz_get_str(
            NULL, 10, x));

    for (;mpz_cmp_si(i,n) < 0; mpz_add_ui(i, i,1))
        {
      fractionDone = (mpz_get_d(i) / (double) n);
      boinc_fraction_done(fractionDone);
      mpz_powm_ui(x, x, 2, fn);

      if (boinc_time_to_checkpoint()) {
        retval = do_checkpoint(&i, &x);
        if (retval) {
          fprintf(
              stderr,
              "%s APP: pepin checkpoint failed ponto 1 "
              "%d\n",
              boinc_msg_prefix(buf, sizeof(buf)),
```

```
            retval);
          exit(retval);
        }
        boinc_checkpoint_completed();
      }

    }
  } else {
    for (mpz_set_ui(i, 1); mpz_cmp_si(i,n) < 0;
        mpz_add_ui(i, i, 1)) {
      fractionDone = (mpz_get_d(i) / (double) n);
      boinc_fraction_done(fractionDone);
      mpz_powm_ui(x, x, 2, fn);

      if (boinc_time_to_checkpoint()) {
        retval = do_checkpoint(&i, &x);
        if (retval) {
          fprintf(
              stderr,
              "%s APP: pepin checkpoint failed ponto 1 "
              "%d\n",
              boinc_msg_prefix(buf, sizeof(buf)),
              retval);
          exit(retval);
        }
        boinc_checkpoint_completed();
      }
    }
  }

  mpz_sub_ui(fk, fn, 1);

  if (mpz_cmp(x, fk) == 0 || k == 0 || k == 1) {
    mpz_clear(x);
    mpz_clear(fk);
    return (1);
  } else {
    mpz_clear(x);
    mpz_clear(fk);
```

```
    return (0);
  }
}

int main(int argc, char **argv) {
  int i, retval;
  int c, n;
  double fd;
  char input_path[512], output_path[512];

  retval = boinc_init();

  if (retval) {
    fprintf(stderr,
        "%s boinc_init returned %d\n",
        boinc_msg_prefix(buf, sizeof(buf)),
        retval);
    exit(retval);
  } else {
    fprintf(stderr, "%s Iniciado com sucesso \n",
        boinc_msg_prefix(buf, sizeof(buf)));
  }

  // open the input file (resolve logical name first)
  //
  boinc_resolve_filename(INPUT_FILENAME,
      input_path, sizeof(input_path));
  infile = boinc_fopen(input_path, "r");
  if (!infile) {
    fprintf(
        stderr,
        "%s Arquivo de entrada nao encontrado %s.\n",
        boinc_msg_prefix(buf, sizeof(buf)),
        input_path);
    exit(-1);
  }

  boinc_resolve_filename(OUTPUT_FILENAME,
      output_path, sizeof(output_path));
```

```
retval = out.open(output_path, "wb");
if (retval) {
  fprintf(
      stderr,
      "%s APP: nao foi possivel criar arquivo "
      "de saida:\n",
      boinc_msg_prefix(buf, sizeof(buf)));
  fprintf(stderr,
      "%s resolved name %s, retval %d\n",
      boinc_msg_prefix(buf, sizeof(buf)),
      output_path, retval);
  perror("open");
  exit(1);
}

int numeroEntrada;

fscanf(infile, "%d", &numeroEntrada);

fprintf(stderr, "%s APP: Lido: %d\n",
    boinc_msg_prefix(buf, sizeof(buf)),
    numeroEntrada);

mpz_t fn;
mpz_init(fn);

i = numeroEntrada;

time_t time1;
time1 = time(NULL);

long expoente = pow(2, i);
mpz_ui_pow_ui(fn, 2, expoente);

mpz_add_ui(fn, fn, 1);

if (pepin(i, fn)) {
  if (mpz_sizeinbase(fn, 10) > 50) {
```

```
      fprintf(
          stderr,
          "%s APP: F %d eh primo de fermat e "
          "possui %d digitos. Demorou %d  \n",
          boinc_msg_prefix(buf, sizeof(buf)), i,
          mpz_sizeinbase(fn, 10), difftime(time(
              NULL), time1));
      out.putchar('P');
    } else {
      fprintf(
          stderr,
          "%s APP: F %d = %s eh primo de fermat. "
          "Demorou %d  \n",
          boinc_msg_prefix(buf, sizeof(buf)), i,
          mpz_get_str(NULL, 10, fn), difftime(
              time(NULL), time1));
      out.putchar('P');
    }
  } else {
    if (mpz_sizeinbase(fn, 10) > 50) {

      fprintf(
          stderr,
          "%s APP: F %d NAO eh primo de fermat e "
          "possui %d digitos. Demorou %d  \n",
          boinc_msg_prefix(buf, sizeof(buf)), i,
          mpz_sizeinbase(fn, 10), difftime(time(
              NULL), time1));
      out.putchar('C');
    } else {
      fprintf(
          stderr,
          "%s APP: F %d = %s eh primo de fermat. "
          "Demorou %d  \n",
          boinc_msg_prefix(buf, sizeof(buf)), i,
          mpz_get_str(NULL, 10, fn), difftime(
              time(NULL), time1));
      out.putchar('C');
    }
```

```
  }

  mpz_clear(fn);

  retval = out.flush();

  if (retval) {
    fprintf(stderr,
        "%s APP: A aplicaÃ§Ã£o falhou! %d\n",
        boinc_msg_prefix(buf, sizeof(buf)),
        retval);
    exit(1);
  }

  boinc_fraction_done(1);
  boinc_finish(0);
}

const char
    *BOINC_RCSID_33ac47a071 =
        "$Id: upper_case.cpp 20315 2010-01-29 "
        "15:50:47Z davea $";
```

ANEXO B ARQUIVO ANDROID.MK

Conteúdo do Arquivo Android.mk

```
# Copyright (C) 2009 The Android Open Source Project
#
# Licensed under the Apache License, Version 2.0
#(the "License"); you may not use this file except
#in compliance with the License. You may obtain a
#copy of the License at
#      http://www.apache.org/licenses/LICENSE-2.0
# Unless required by applicable law or agreed to in
#writing, software distributed under the License is
#distributed on an "AS IS" BASIS, WITHOUT WARRANTIES
#OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing
#permissions and limitations under the License.
#
LOCAL_PATH := $(call my-dir)
NDK_ROOT := "/home/user/android-ndk"
WORKSPACE := "/home/user/workspace"

LOCAL_LDLIBS += -L$(WORKSPACE)/ssl -lssl -lcrypto

#LIBS
include $(CLEAR_VARS)
LOCAL_MODULE    := libboinclib
LOCAL_SRC_FILES := libboinclib.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE    := libgnustl_shared
```

```
LOCAL_SRC_FILES := libgnustl_shared.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := libssl
LOCAL_SRC_FILES := libssl.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE := libcrypto
LOCAL_SRC_FILES := libcrypto.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE    := libboincapi
LOCAL_SRC_FILES := libboincapi.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE    := libnativeboinc_utils
LOCAL_SRC_FILES := libnativeboinc_utils.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_MODULE    := libgmp
LOCAL_SRC_FILES := libgmp.so
include $(PREBUILT_SHARED_LIBRARY)

include $(CLEAR_VARS)
LOCAL_LDLIBS := -llog -ldl -lz
LOCAL_MODULE    := pepin
LOCAL_SRC_FILES := pepin.cpp

LOCAL_SHARED_LIBRARIES := libssl \
libcrypto \
libboinclib \
libboincapi \
libgnustl_shared \
libnativeboinc_utils \
```

libgmp

```
LOCAL_C_INCLUDES := $(LOCAL_PATH) \
$(WORKSPACE)/Bibliotecas/arm/lib \
$(WORKSPACE)/Bibliotecas/arm/api \
$(NDK_ROOT)/sources/cxx-stl/gnu-libstdc++/4.6/include \
$(NDK_ROOT)/sources/cxx-stl/gnu-libstdc++/4.6/libs/armeabi-v7a/include \
$(LOCAL_PATH)/jni

include $(BUILD_EXECUTABLE)
```

ANEXO C ARQUIVO APP_INFO.XML

Conteúdo do Arquivo app_info.xml

```
<app_info>
<app>
    <name>pepin_app</name>
    <user_friendly_name>Verifica a primalidade de numeros
        de Fermat</user_friendly_name>
</app>

<file_info>
    <name>pepin_6.48_arm-android-linux-gnu</name>
    <executable/>
</file_info>

<file_info>
    <name>libboincapi.so</name>
    <executable/>
</file_info>

<file_info>
    <name>libboinclib.so</name>
    <executable/>
</file_info>

<file_info>
    <name>libcrypto.so</name>
    <executable/>
</file_info>

<file_info>
```

```
    <name>libgmp.so</name>
    <executable/>
</file_info>

<file_info>
    <name>libgnustl_shared.so</name>
    <executable/>
</file_info>

<file_info>
    <name>libnativeboinc_utils.so</name>
    <executable/>
</file_info>

<file_info>
    <name>libssl.so</name>
    <executable/>
</file_info>

<app_version>
    <app_name>pepin_app</app_name>
    <version_num>648</version_num>
    <api_version>6.12.38</api_version>
    <file_ref>
        <file_name>
                pepin_6.48_arm-android-linux-gnu
            </file_name>
        <main_program/>
    </file_ref>

    <file_ref>
        <file_name>libboincapi.so</file_name>
        <open_name>libboincapi.so</open_name>
    </file_ref>

    <file_ref>
        <file_name>libboinclib.so</file_name>
        <open_name>libboinclib.so</open_name>
    </file_ref>
```

```
    <file_ref>
        <file_name>libcrypto.so</file_name>
        <open_name>libcrypto.so</open_name>
    </file_ref>

    <file_ref>
        <file_name>libgmp.so</file_name>
        <open_name>libgmp.so</open_name>
    </file_ref>

    <file_ref>
        <file_name>libgnustl_shared.so</file_name>
        <open_name>libgnustl_shared.so</open_name>
    </file_ref>

    <file_ref>
        <file_name>libnativeboinc_utils.so</file_name>
        <open_name>libnativeboinc_utils.so</open_name>
    </file_ref>

    <file_ref>
        <file_name>libssl.so</file_name>
        <open_name>libssl.so</open_name>
    </file_ref>

    <platform>arm-android-linux-gnu</platform>
    <avg_ncpus>1.000000</avg_ncpus>
    <max_ncpus>1.000000</max_ncpus>
</app_version>
</app_info>
```