UNIVERSIDADE DE CAXIAS DO SUL
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

MAUREN RIBEIRO BERTI

Uso de computação pública para o cálculo dos dígitos do Pi

André Luís Martinotto
Orientador

Caxias do Sul, Julho de 2015

Uso de computação pública para o cálculo dos dígitos do Pi

por

Mauren Ribeiro Berti

Projeto de Diplomação submetido ao curso de Bacharelado em Ciência da Computação do Centro de Ciências Exatas e Tecnologia da Universidade de Caxias do Sul, como requisito obrigatório para graduação.

Projeto de Diplomação

Orientador: André Luís Martinotto
Banca examinadora:

Helena Grazziotin Ribeiro
CCET/UCS

Scheila de Ávila e Silva
CCET/UCS

Projeto de Diplomação apresentado em
06 de julho de 2015

Daniel Luís Notari
Coordenador

SUMÁRIO

LISTA DE FIGURAS 5

LISTA DE TABELAS 6

LISTA DE QUADROS 7

RESUMO 8

ABSTRACT 9

1 INTRODUÇÃO 10
1.1 Objetivo 12
1.2 Estrutura do Trabalho 13

2 MÉTODOS DE CÁLCULO DO PI 14
2.1 Método de Arquimedes 14
2.2 Método de Monte Carlo 15
2.3 Método da Integração 16
2.4 Métodos com séries 17
2.4.1 Série de Taylor 17
2.4.2 Série de Leibniz 17
2.4.3 Série de Euler 18
2.4.4 Série de Gauss 18
2.5 Algoritmos de Extração de Dígitos 19
2.5.1 Fórmula de Bailey-Borwein-Plouffe 19
2.5.2 Fórmula de Bellard 21

3 COMPUTAÇÃO PÚBLICA 23
3.1 Plataforma BOINC 24
3.1.1 Servidor BOINC 26
3.1.2 Cliente BOINC 28

3.1.3 Aplicação 30

4 ESTUDO DE CASO 31
4.1 Servidor BOINC 32
4.1.1 Configuração do servidor 32
4.1.2 Criação das WUs 33
4.2 Aplicação 37
4.2.1 Compilação da aplicação 39

5 RESULTADOS OBTIDOS 40
5.1 Validação da aplicação 40
5.2 Teste de desempenho 41

6 CONSIDERAÇÕES FINAIS 43
6.1 Trabalhos Futuros 44

REFERÊNCIAS 45

ANEXO A - CÓDIGO-FONTE DO PROJETO 48
7.1 Aplicação principal 48
7.1.1 Arquivo de compilação para Linux 64-bit 53
7.1.2 Arquivo de compilação para Android ARM 54
7.2 Assimilador de resultados 54
7.2.1 Arquivo de compilação do assimilador para Linux 64-bit 56
7.3 Gerador de *workunits* 57
7.3.1 Arquivo de compilação do gerador de *workunits* para Linux 64-bit . . 59

LISTA DE FIGURAS

2.1 Demonstração do cálculo de Pi com polígonos, utilizando polígonos com 6, 12 e 24 lados. 14
2.2 Um quadrado circunscrito ao círculo unitário. 15
2.3 Simulação de Monte Carlo para o cálculo do Pi com 1000 pontos. 16
2.4 Gráfico da função $f(x) = \frac{4}{1+x^2}$ no intervalo $[0, 1]$ e exemplo de estimativa da área sob a curva. 16
2.5 Ilustração do círculo trigonométrico. 18

3.1 Arquitetura BOINC (SMIDERLE, 2012). 25
3.2 Funcionamento do sistema de recompensas do BOINC (BOINC Documentation, 2015). 26
3.3 Servidor BOINC e sua interação com o cliente (SMIDERLE, 2012). 27
3.4 Configuração das opções do cliente BOINC. 29
3.5 Arquitetura do cliente BOINC (SMIDERLE, 2012). 29

4.1 Estrutura de pastas das aplicações do projeto BOINC. 31
4.2 Tela de gerenciamento do servidor BOINC. 36

LISTA DE TABELAS

5.1 Listagem de alguns valores hexadecimais do Pi calculados pela aplicação. 41

5.2 Computadores utilizados para tabular o desempenho do aplicativo com configuração e sistema operacional. 41

5.3 Resultados do processamento da *workunit* para o cálculo do dígito 100.000 do Pi. 41

LISTA DE QUADROS

4.1 Arquivo `version.xml`. 32
4.2 *Template* de entrada `input.xml`. 33
4.3 *Template* de saída `output.xml`. 34
4.4 Exemplo do arquivo de saída do assimilador. 37
4.5 Configuração do *daemon* do assimilador no arquivo `config.xml`. . . 37
4.6 Exemplo do arquivo de entrada para o cálculo do dígito 5000. 38
4.7 Função para a leitura do arquivo que contém a posição do dígito do Pi que será calculado. 38
7.1 bbp_pi.h - definições das funções da aplicação. 48
7.2 bbp_pi.cpp - implementação das funções da aplicação. 49
7.3 prc_pi.cpp - ponto de entrada da aplicação. 51
7.4 Makefile.app 53
7.5 Makefile.droid 54
7.6 Implementação do assimilador. 54
7.7 Makefile.assimilator 56
7.8 Implementação do gerador de *workunits*. 57
7.9 Makefile.wugen 59

RESUMO

A constante Pi é a constante matemática mais antiga conhecida. Apesar da antiguidade do nosso conhecimento do Pi, este ainda é objeto de pesquisa em diversas áreas. De fato, suas propriedades seguem sendo investigadas, e novos e mais poderosos métodos continuam sendo desenvolvidos para o cálculo de seus dígitos. O cálculo dos dígitos do Pi é uma tarefa computacionalmente intensiva, o que justifica a utilização de uma plataforma de computação pública para esta finalidade.

Neste trabalho, foi abordado o cálculo dos dígitos do Pi utilizando uma plataforma de computação pública. Foram pesquisados métodos de cálculo dos dígitos do Pi. Entre os métodos estudados, um método de extração de dígitos foi selecionado e implementado. Esta implementação foi desenvolvida em linguagem de programação C, utilizando a plataforma BOINC e a biblioteca de manipulação de números com precisão arbitrária GNU MPFR, devido à necessidade de precisão arbitrária nos cálculos com ponto-flutuante.

A aplicação desenvolvida foi submetida a testes de validação e medições de desempenho. Para a validação, foram calculados 10000 dígitos do Pi e os valores obtidos foram validados utilizando os dados fornecidos pelo projeto HexPi. Para as medições de desempenho, o dígito de número 100.000 foi calculado nas plataformas Linux, Windows e Android, sendo que o melhor desempenho foi obtido na plataforma Linux. O desempenho na plataforma Android foi superior ao do Windows mesmo quando executado em um hardware de desempenho inferior.

Palavras-chave: Computação pública, BOINC, computação distribuída, Pi.

Use of public computing to calculate the digits of Pi

ABSTRACT

Pi is the oldest known mathematical constant. Despite the age of our knowledge of Pi, it still stays among actively researched objects in many fields of study. Its properties remain being investigated and new and more powerful methods keep being developed for Pi digit computation. The calculation of Pi digits is an example of compute-intensive task, and this fact justifies the use of a public-resource computing platform for this purpose.

This project described the calculation of Pi digits using a public-resource computing platform. Methods of Pi digit calculation were researched. Among the researched methods, a digit-extraction algorithm was selected and implementead. This implementation was developed in C programming language, using the public-resource computing platform BOINC and the arbitrary-precision number manipulation library GNU MPFR, due to the need of arbitrary precision math for floating-point calculations.

The developed application was submitted to validation tests and performance measures. For validation, 10000 Pi digits were calculated and the resulting values were validated using data provided by HexPi project. For performance measures, the 100.000th digit of Pi was calculated in Linux, Windows and Android platforms. The application had its best performance under Linux platform. The performance for Android platform was superior to Windows, even when running under a hardware with worse performance.

Keywords: public computing,BOINC,distributed computing,Pi.

1 INTRODUÇÃO

Aplicações que demandam um alto poder de processamento, ou computacionalmente intensivas, são aquelas que realizam um grande volume de cálculos, ou ainda, cálculos sobre um conjunto relativamente grande de dados (WANG et al., 2009). Entre aplicações que demandam um alto poder computacional encontram-se as aplicações das áreas de mineração de dados, bioinformática, previsão climática, modelos atmosféricos e cálculos moleculares (PARSONS; OJA, 2014).

Uma alternativa para suprir essa necessidade de alto processamento consiste na utilização do paradigma de computação pública, que é uma forma de utilizar o tempo ocioso de computadores e videogames, distribuídos ao redor do mundo, para realizar cálculos computacionais que de outra forma só seriam possíveis de serem realizados utilizando um supercomputador. Além disso, este paradigma encoraja a consciência pública sobre pesquisas científicas atuais, criando comunidades ao redor de interesses científicos e mantendo o público informado sobre as descobertas científicas realizadas (ANDERSON, 2004).

Com o aumento do poder computacional dos dispositivos móveis, esses tornaram-se uma arquitetura atrativa para a execução de aplicações de computação pública. De fato, já existem projetos que utilizam o tempo ocioso destes dispositivos para a realização de processos computacionalmente intensivos. Por exemplo, a plataforma de computação pública BOINC (*Berkeley Open Infrastructure for Network Computing*) possui uma implementação para dispositivos móveis que utilizem o sistema operacional Android, de forma que os projetos desenvolvidos para a plataforma BOINC possam se beneficiar do poder computacional destes dispositivos (BOINC Documentation, 2015).

Atualmente, existem cerca de 41 projetos utilizando a plataforma BOINC. Alguns destes projetos são: o SETI@Home, que busca por vida extraterrestre inteligente (SETI@Home, 2015); o ABC@Home, que procura por conjuntos de números que atendam aos requisitos da conjectura ABC (ABC@Home, 2015); o LHC@Home, que executa simulações vinculadas ao colisor de hádrons do CERN (LHC@Home, 2015); e o PrimeGrid, que busca incessantemente por números primos e cujo objetivo

é calcular o maior número primo conhecido (PrimeGrid, 2015).

Além disso, existem projetos de computação pública que não utilizam a plataforma BOINC como, por exemplo: o projeto Folding@Home, que realiza simulações com o intuito de buscar a cura para doenças causadas pelo enovelamento de proteínas (Folding@Home, 2015); o Background Pi, que calcula dígitos decimais do Pi de forma distribuída (Background Pi, 2015); e o Twin Prime Search, que busca por números primos gêmeos (Twin Prime Search, 2015).

Entre os problemas matemáticos computacionalmente intensivos encontra-se o cálculo dos dígitos decimais do Pi. O número Pi é um dos objetos matemáticos mais instigantes existentes, sendo a mais antiga constante matemática conhecida, cujo valor é a razão entre a circunferência e o diâmetro do círculo. Apesar da antiguidade do nosso conhecimento sobre o Pi, ele ainda é fonte de pesquisas em diversas áreas. De fato, suas propriedades seguem sendo investigadas, sendo criados novos e mais poderosos métodos matemáticos para o cálculo do seu valor (SILVEIRA, 2001).

Uma das maiores dificuldades no cálculo dos dígitos do Pi consiste no fato deste não ser uma fração. Se o Pi pudesse ser escrito como uma fração $\frac{m}{n}$, seu cálculo poderia ser resumido na busca dos valores inteiros m e n, ou em explorar a periodicidade de sua representação decimal. Porém, a irracionalidade do Pi, por si só, não é suficiente para determinar a dificuldade do seu cálculo. O Pi é difícil de ser calculado porque, além de ser irracional, é imprevisível, ou seja, seus algarismos distribuem-se de forma aleatória (SILVEIRA, 2001).

Apesar da maioria dos cálculos práticos não utilizarem aproximações do Pi maiores do que os valores $3,14$ e $3,1416$, com exceção daqueles cálculos trigonométricos que necessitam de alta precisão, o cálculo do Pi com um número cada vez maior de casas decimais se justifica por alguns fatores (SILVEIRA, 2001):

- desejo de enfrentar um problema matemático difícil;
- desejo de entrar para a História da Matemática, com um recorde de cálculo dos dígitos do Pi;
- possibilidade de demonstrar o potencial de novos métodos de cálculo;
- oportunidade de estudar a distribuição dos seus dígitos;
- possibilidade de demonstrar o desempenho dos computadores atuais.

O cálculo dos dígitos do Pi já foi explorado em aplicações de computação pública. Entre os projetos já desenvolvidos, estão o projeto PiHex (PiHex Project, 2015) e o projeto Background Pi (Background Pi, 2015), que foi anteriormente citado. Ambos os projetos foram desenvolvidos sem auxílio da plataforma BOINC.

O Background Pi foi implementado como uma aplicação para o sistema operacional Windows. Esse foi escrito em linguagem C++ e calcula 9 dígitos do Pi a cada execução. Há pouca informação sobre o algoritmo utilizado, sobre seu funci-

onamento e sobre o estado atual do projeto (Background Pi, 2015). Já o projeto PiHex foi finalizado em 1998, tendo sido calculados os dígitos binários do Pi de forma distribuída até a quadrilionésima posição. Este projeto foi desenvolvido dentro da Universidade Simon Fraser, no Canadá, e foi escrito em linguagem C (PiHex Project, 2015).

Apesar de todo o conhecimento e interesse a respeito do cálculo dos algarismos de Pi, não há nenhum projeto de computação pública que possibilite a utilização de dispositivos móveis para este propósito. Assim, torna-se interessante o desenvolvimento de um projeto de computação pública que permita a utilização desse tipo de dispositivo, juntamente com a execução em computadores pessoais.

Desta forma, neste trabalho foi desenvolvido um projeto que possibilita a utilização do poder computacional de dispositivos móveis e de computadores pessoais para o cálculo do Pi. Para o desenvolvimento desse projeto foi utilizada a plataforma BOINC, devido ao seu suporte a diferentes arquiteturas e sistemas operacionais (BOINC Documentation, 2015). Além disso, foi utilizada a biblioteca MPFR (*GNU Multi Precision Floating-Point Reliable Library*), que é um aprimoramento da biblioteca GMP (*GNU Mathematical Precision*). Esta é uma biblioteca de código aberto que oferece estruturas de dados e rotinas para trabalhar com números de precisão arbitrária, sejam eles inteiros, fracionários ou de ponto flutuante (GRANLUND, 2002).

1.1 Objetivo

O objetivo deste trabalho consistiu no desenvolvimento e análise de desempenho de um sistema que calcula os dígitos do número Pi de forma distribuída. Este sistema foi desenvolvido utilizando a plataforma BOINC e encontra-se disponível para dispositivos móveis e computadores pessoais.

De modo a atingir este objetivo, foram atingidos os seguintes objetivos específicos:

- Desenvolvimento da aplicação para plataformas as Windows, Linux e Android;
- Avaliação do desempenho da aplicação nas diferentes plataformas.

1.2 Estrutura do Trabalho

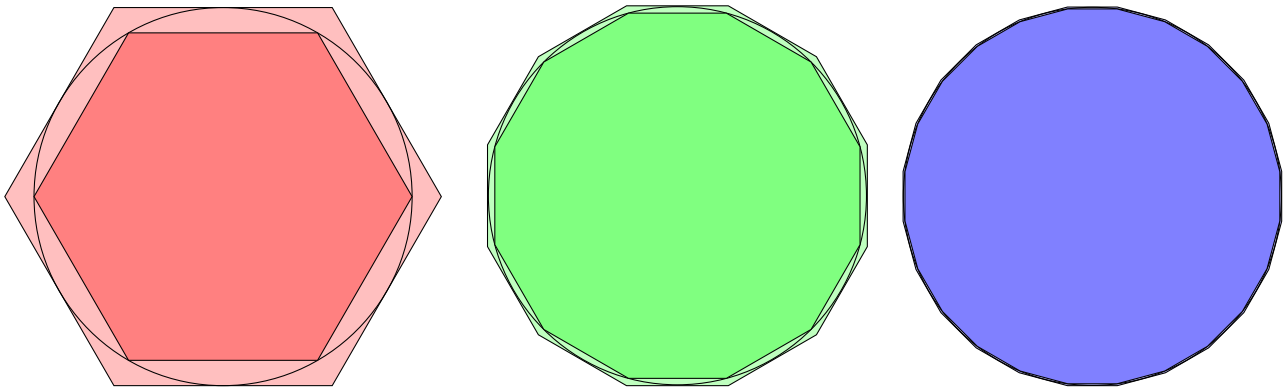
Este trabalho está dividido em 6 capítulos. No Capítulo 2 são descritos alguns métodos para o cálculo do Pi. No Capítulo 3 são abordados conceitos relativos à computação pública e ao funcionamento da plataforma BOINC. No Capítulo 4, a aplicação desenvolvida é apresentada e são abordadas as técnicas utilizadas para seu desenvolvimento. No Capítulo 5, são apresentados os resultados obtidos com o desenvolvimento desta aplicação. Por fim, no Capítulo 6, são apresentadas as considerações finais e algumas sugestões de trabalhos futuros.

2 MÉTODOS DE CÁLCULO DO PI

Apesar da antiguidade do conhecimento a respeito do Pi, estudos a respeito desta constante e esforços para calcular um número maior de dígitos permanecem sendo realizados até hoje. O valor do Pi é a razão entre a circunferência e o diâmetro de qualquer círculo. Além desta forma clássica para obtermos o valor desta constante, existem inúmeras outras, sendo que algumas destas formas serão abordadas neste capítulo. Estes métodos incluem cálculos comuns baseados em trigonometria, cálculos baseados em séries, métodos estatísticos e algoritmos específicos para extração de dígitos. Um conjunto mais abrangente de métodos para o cálculo do valor da constante Pi pode ser encontrado no artigo *Pi Formulas* (WEISSTEIN, 2015).

2.1 Método de Arquimedes

Este método consiste em utilizar um círculo possuindo um polígono inscrito e outro circunscrito, ambos com um número n de lados, tal que $n = 6 \cdot 2^k$, com $k \in \mathbb{N}$ (Figura 2.1).



(a) (b) (c)

Figura 2.1: Demonstração do cálculo de Pi com polígonos, utilizando polígonos com 6, 12 e 24 lados.

Considerando que $2P_1$ seja o perímetro do polígono inscrito, $2P_2$ seja o perímetro

do polígono circunscrito, C seja a medida da circunferência e considerando r como o raio do círculo, é possível verificar que $\frac{2P_1}{2r} < \frac{C}{2r} < \frac{2P_2}{2r}$. Observa-se ainda que, com o aumento do número de lados do polígono (Figuras 2.1b e 2.1c), mais próxima do valor de π fica a razão $\frac{C}{2r}$ (SHENGDA et al., 2012; MIRANDA, 2015).

Este método foi utilizado por Arquimedes para aproximar o valor do Pi utilizando polígonos de 96 lados, para o qual ele conseguiu obter os valores de $3\frac{10}{71} < \pi < 3\frac{1}{7}$, ou seja, $3.14084507 < \pi < 3.14285714$ (MCKEEMAN, 2010). Assim, calculando-se a média desses dois valores, obtém-se um valor para Pi de 3.14185110, ou seja, tem-se uma aproximação grosseira do valor exato do Pi [1]. Uma descrição mais detalhada do método de Arquimedes encontra-se no artigo *The Computation of Pi by Archimedes* (MCKEEMAN, 2010).

2.2 Método de Monte Carlo

Neste método, considera-se um quadrado circunscrito a um círculo unitário, ou seja, um círculo cujo raio tem comprimento de 1 unidade, conforme pode ser observado na Figura 2.2.

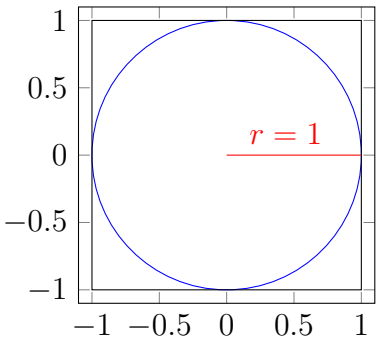


Figura 2.2: Um quadrado circunscrito ao círculo unitário.

Em seguida, gera-se um conjunto n de pontos aleatórios $\{(x, y)\}_{i=1}^{n}$ localizados no interior do quadrado unitário. Então, determina-se a razão $\rho = \frac{m}{n}$, onde n é o número de pontos gerados e m é o número de pontos que foram gerados no interior do círculo, ou seja, que atendam à condição $x_i^2 + y_i^2 \leq 1$. Esta razão é uma aproximação de $\frac{\pi}{4}$ (Monte Carlo Pi, 2015). É importante observar que quanto maior for o número de pontos gerados, melhor será a aproximação do valor do Pi.

Na Figura 2.3 tem-se um exemplo da aplicação do método de Monte Carlo com a geração de 1000 pontos, sendo que destes, 787 atenderam à condição $x_i^2 + y_i^2 \leq 1$. Desta forma, obtemos:

$$\rho = \frac{m}{n} = \frac{787}{1000} = 0.787 \tag{2.1}$$

[1] O valor de Pi com 15 casas decimais é de 3.141592653589793.

$$\pi \approx \rho \cdot 4 = 0.787 \cdot 4 = 3.148. \tag{2.2}$$

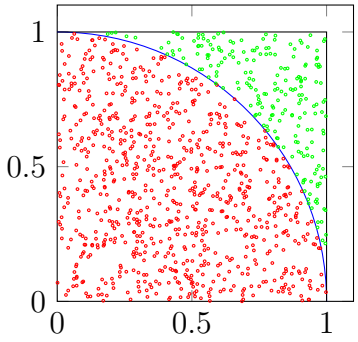


Figura 2.3: Simulação de Monte Carlo para o cálculo do Pi com 1000 pontos.

2.3 Método da Integração

O método da integração consiste em calcular a integral da função $f(x) = \frac{4}{1+x^2}$ no intervalo $[0, 1]$. Uma aproximação desta integral pode ser calculada utilizando retângulos para aproximar a área abaixo do gráfico da função $f(x)$. Uma ilustração de como funcionaria o cálculo desta área encontra-se na Figura 2.4b.

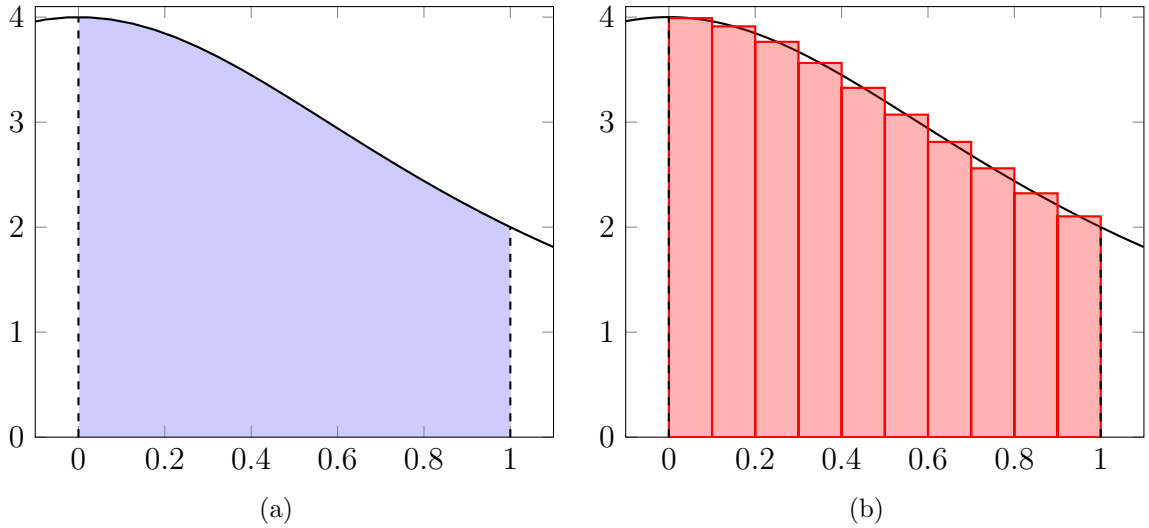


Figura 2.4: Gráfico da função $f(x) = \frac{4}{1+x^2}$ no intervalo $[0, 1]$ e exemplo de estimativa da área sob a curva.

Se calcularmos a área de um número suficientemente grande de retângulos, ao somarmos estes valores, obteremos um resultado bastante próximo do valor do Pi. Por exemplo, para um milhão de retângulos, teríamos uma estimativa do valor do Pi de 3.14159165359.

2.4 Métodos com séries

Nesta seção, serão apresentadas algumas séries utilizadas para o cálculo do valor do Pi. Comparativamente a outros métodos, estas séries não são tão eficientes, em função de necessitarem de um número muito grande de termos para convergirem, ou ainda, por utilizarem cálculos computacionalmente custosos.

2.4.1 Série de Taylor

Podemos expandir a série de Taylor para a função $f(x) = \frac{1}{1+x^2}$, conforme a Equação 2.3,

$$\sum_{n=0}^{\infty} 1 - x^2 + x^4 - x^6 + x^8 - ... = \sum_{n=0}^{\infty} (-1)^n x^{2n}, \tag{2.3}$$

onde n é o índice do termo. Esta série, quando integrada no intervalo $[0, 1]$, resulta no valor de $\frac{\pi}{4}$. Já quando integrada no intervalo $[0, \infty)$, tem-se o valor de $\frac{\pi}{2}$ (SHENGDA et al., 2012).

2.4.2 Série de Leibniz

A série de Leibniz, que permite o cálculo da arcotangente de 1, é descrita como

$$\frac{\pi}{4} = \arctan(1) = \sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1}, \tag{2.4}$$

onde n representa o índice do termo.

A arcotangente, ou tangente inversa, é uma função trigonométrica que indica, em radianos, o ângulo a que um determinado valor de tangente se refere. Considerando que, em radianos, o tamanho do círculo trigonométrico seja de 2π, conforme ilustrado na Figura 2.5, o valor de $\arctan(1)$ será aproximadamente $\frac{\pi}{4}$, pois o ângulo para o qual $\tan(x) = 1$ é $45°$, ou seja, $\frac{\pi}{4}$ (SHENGDA et al., 2012; WEISSTEIN, 2002).

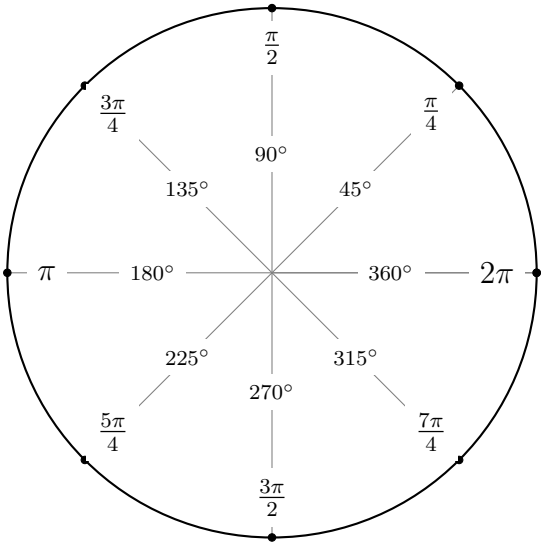


Figura 2.5: Ilustração do círculo trigonométrico.

2.4.3 Série de Euler

De forma similar à série de Leibniz, a série de Euler permite o cálculo do valor de $\frac{\pi}{4}$, porém utilizando o valor da soma das arcotangentes de $\frac{1}{2}$ e $\frac{1}{3}$, ou seja,

$$\begin{aligned}\frac{\pi}{4} &= \arctan\left(\frac{1}{3}\right) + \arctan\left(\frac{1}{2}\right) \\ &= \left(\sum_{n=0}^{\infty} \frac{(-1)^n (\frac{1}{3})^{2n+1}}{2n+1} + \sum_{n=0}^{\infty} \frac{(-1)^n (\frac{1}{2})^{2n+1}}{2n+1}\right),\end{aligned} \tag{2.5}$$

onde n é o índice do termo.

O uso da soma arcotangentes de $\frac{1}{2}$ e $\frac{1}{3}$ se justifica pelo fato desta soma resultar em uma aproximação de $\frac{\pi}{4}$, que representa 45° no círculo trigonométrico (SMITH, 2010; SHENGDA et al., 2012).

2.4.4 Série de Gauss

Utilizando a soma das arcotangentes de $\frac{1}{18}$, $\frac{1}{57}$ e $\frac{1}{239}$, a série de Gauss produz o valor de Pi de acordo com a Equação 2.6,

$$\begin{aligned}\pi &= 48\arctan\left(\frac{1}{18}\right) + 32\arctan\left(\frac{1}{57}\right) - 20\arctan\left(\frac{1}{239}\right) \\ &= 48\sum_{n=0}^{\infty} \frac{(-1)^n (\frac{1}{18})^{2n+1}}{2n+1} + 32\sum_{n=0}^{\infty} \frac{(-1)^n (\frac{1}{57})^{2n+1}}{2n+1} - 20\sum_{n=0}^{\infty} \frac{(-1)^n (\frac{1}{239})^{2n+1}}{2n+1},\end{aligned} \tag{2.6}$$

onde n é o índice do termo (SMITH, 2010; SHENGDA et al., 2012).

2.5 Algoritmos de Extração de Dígitos

Algoritmos de extração de dígitos são métodos que permitem o cálculo de um dígito em uma posição arbitrária sem a necessidade do cálculo dos dígitos predecessores. Existem fórmulas deste tipo para o cálculo de várias constantes matemáticas. Nesta seção, serão abordados dois métodos que possibilitam este tipo de cálculo dos dígitos do Pi (WEISSTEIN, 2015).

2.5.1 Fórmula de Bailey-Borwein-Plouffe

Na Equação 2.7, temos a fórmula de Bailey-Borwein-Plouffe (ou BBP), que foi desenvolvida por David H. Bailey, Peter B. Borwein e Simon Plouffe em 1996.

$$\pi = \frac{1}{16^n} \sum_{n=0}^{\infty} \left(\frac{4}{8n+1} - \frac{2}{8n+4} - \frac{1}{8n+5} - \frac{1}{8n+6} \right), \tag{2.7}$$

onde n é o índice do termo na série.

A Equação 2.7 permite calcular o Pi em tempo razoavelmente curto com qualquer precisão, embora não seja tão eficiente quanto outras fórmulas que são utilizadas atualmente. Porém, esta fórmula possui uma propriedade interessante, uma vez que com algumas modificações, esta equação permite o cálculo de qualquer dígito hexadecimal ou binário do Pi a partir de uma posição arbitrária (BAILEY, 2006).

Para extrairmos dígitos em posições arbitrárias, a equação BBP pode ser reescrita no formato

$$\{16^d \pi\} = \{4\{16^d S_1\} - 2\{16^d S_4\} - \{16^d S_5\} - \{16^d S_6\}\}, \tag{2.8}$$

em que d é a posição do dígito que se deseja calcular (sendo $d = 0$ o primeiro dígito hexadecimal). Já $\{\cdot\}$ denota a extração da parte decimal do número. Por fim, o termo S_j é descrito por

$$S_j = \sum_{n=0}^{\infty} \frac{1}{16^n (8n+j)}, \tag{2.9}$$

onde n é o índice do termo na série e j é um valor inteiro de modo que $j \in \{1, 4, 5, 6\}$. Para fins de aplicação na Equação 2.8, S_j deverá ser expandido como

$$\begin{aligned} \{16^d S_j\} &= \left\{ \left\{ \sum_{n=0}^{d} \frac{16^{d-n}}{8n+j} \right\} + \sum_{n=d+1}^{\infty} \frac{16^{d-n}}{8n+j} \right\} \\ &= \left\{ \left\{ \sum_{n=0}^{d} \frac{16^{d-n} \bmod k}{8n+j} \right\} + \sum_{n=d+1}^{\infty} \frac{16^{d-n}}{8n+j} \right\}, \end{aligned} \tag{2.10}$$

onde n é o índice do termo na fórmula, d é o dígito que está sendo calculado e,

como já mencionado, j é o valor inteiro ($j \in \{1, 4, 5, 6\}$) a ser somado ao termo $8n$ do denominador. Substituindo-se na Equação 2.8 os termos S_j, temos a seguinte Equação

$$\begin{aligned}
\left\{16^d\pi\right\} &= \left\{4\left\{16^dS_1\right\} - 2\left\{16^dS_4\right\} - \left\{16^dS_5\right\} - \left\{16^dS_6\right\}\right\} \\
&= \left\{4\left\{\left\{\sum_{n=0}^{d}\frac{16^{d-n}\bmod n}{8n+1}\right\} + \sum_{n=d+1}^{\infty}\frac{16^{d-n}}{8n+1}\right\}\right\} \\
&- \left\{2\left\{\left\{\sum_{n=0}^{d}\frac{16^{d-n}\bmod n}{8n+4}\right\} + \sum_{n=d+1}^{\infty}\frac{16^{d-n}}{8n+4}\right\}\right\} \\
&- \left\{\left\{\left\{\sum_{n=0}^{d}\frac{16^{d-n}\bmod n}{8n+5}\right\} + \sum_{n=d+1}^{\infty}\frac{16^{d-n}}{8n+5}\right\}\right\} \\
&- \left\{\left\{\left\{\sum_{n=0}^{d}\frac{16^{d-n}\bmod n}{8n+6}\right\} + \sum_{n=d+1}^{\infty}\frac{16^{d-n}}{8n+6}\right\}\right\}
\end{aligned} \tag{2.11}$$

A inserção de $\bmod n$ no numerador da fração justifica-se pelo fato de que somente a parte fracionária do primeiro somatório interessa para este cálculo. Como o segundo somatório aproxima-se rapidamente de zero, apenas um pequeno número de termos precisa ser calculado. Já o valor de $16^{d-n} \bmod n$ é calculado pelo Algoritmo 1.

Algoritmo 1: Algoritmo para o cálculo da exponenciação módulo n.

entrada: base numérica inteira b, expoente inteiro m, divisor inteiro n
saída : $r = b^m \bmod n$

1 $r \leftarrow 1.0$;
2 $t \leftarrow$ maior potência de 2 tal que $t \leq m$;
3 **se** $n \geq t$ **então**
4 $\quad r \leftarrow br \bmod n$;
5 $\quad n \leftarrow m - t$;
6 **fimse**
7 $t \leftarrow t/2$
8 **se** $t \geq 1$ **então**
9 $\quad r \leftarrow r^2 \bmod n$;
10 $\quad$ **vá para** 3;
11 **fimse**

Após o cálculo do dígito, conforme descrito na Equação 2.11, o valor obtido deve ser transformado em um valor no intervalo $[0, 1)$, conforme Equação 2.12.

$$\left\{16^d\pi\right\} = v - \lfloor(v - \lfloor v\rfloor + 1)\rfloor, \tag{2.12}$$

em que v é o valor resultante da Equação 2.8 e o operador $\lfloor \cdot \rfloor$ denota a função piso[2].

Sobre o valor decimal obtido para o dígito, será aplicada a transformação para base hexadecimal. Para converter o valor decimal obtido em dígitos hexadecimais, utiliza-se o Algoritmo 2.

Algoritmo 2: Algoritmo para mudança da base decimal para hexadecimal até 10 dígitos hexadecimais.

entrada: valor decimal d
saída : valor hexadecimal h

```
1 h ← "";
2 a ← v;
3 enquanto tamanho(h) < 10
4 |   a ← a · 16;
5 |   d ← piso(a);
6 |   h ← h + digito_hexa(d);
7 |   a ← a − piso(a);
8 fimenquanto
```

Em aritmética de ponto flutuante de 64-bit, este método permite obter até 9 dígitos hexadecimais, enquanto que utilizando aritmética de ponto flutuante de 128-bit obtém-se 24 dígitos hexadecimais (BAILEY, 2006).

2.5.2 Fórmula de Bellard

A fórmula de Bellard é uma série baseada na fórmula BBP, sendo que esta fórmula permite que dígitos binários ou hexadecimais do Pi sejam calculados com um ganho de performance de aproximadamente 43%, quando comparada à fórmula BBP. Na Equação 2.13 tem-se a fórmula de Bellard, onde n corresponde ao índice do termo.

$$\pi = \frac{1}{2^6} \sum_{n=0}^{\infty} \frac{(-1)^n}{2^{10n}} \left(-\frac{2^5}{4n+1} - \frac{1}{4n+3} + \frac{2^8}{10n+1} - \frac{2^6}{10n+3} - \frac{2^2}{10n+5} - \frac{2^2}{10n+7} + \frac{1}{10n+9} \right) \tag{2.13}$$

Assim como a fórmula BBP, que foi descrita na Seção 2.5.1, a fórmula de Bellard pode ser manipulada de modo a extrair os dígitos do Pi (BELLARD, 1997). De fato, a Equação 2.13 pode ser generalizada como

$$\pi = \sum_{k=0}^{\infty} \left(\frac{1}{16} \right)^k \left(\frac{4+8r}{8k+1} - \frac{8r}{8k+2} - \frac{4r}{8k+3} - \frac{2+8r}{8k+4} - \frac{1+2r}{8k+5} - \frac{1+2r}{8k+6} + \frac{r}{8k+7} \right) \tag{2.14}$$

para qualquer valor complexo de r, de modo que esta fórmula se iguala à Equação

[2] A função piso $f(x) = \lfloor x \rfloor$ é uma função matemática que retorna o maior valor inteiro que seja menor ou igual ao valor de x (LIMA, 2015).

2.7 para o caso especial $r = 0$ (WEISSTEIN, 2002). Assim, sobre a Equação 2.14 pode ser aplicado o mesmo processo que foi descrito na Seção 2.5.1, e obtém-se o valor de qualquer dígito do Pi em base hexadecimal.

3 COMPUTAÇÃO PÚBLICA

Computação pública, também conhecida como computação global ou computação voluntária, é um paradigma de processamento que utiliza dispositivos oferecidos pelo público em geral, conectados à internet, para a realização de tarefas computacionalmente intensivas, que de outra forma só seriam viáveis utilizando o poder computacional de supercomputadores (ANDERSON; FEDAK, 2006).

A utilização da computação pública vem crescendo devido principalmente ao aumento do poder de processamento dos computadores pessoais nas últimas décadas, o que viabilizou a utilização destes dispositivos para o processamento de alto desempenho. De fato, a computação pública é uma alternativa ao uso de centros de supercomputação em situações que não necessitem de um controle tão rígido da execução da aplicação e em que as tarefas não necessitem de comunicação. Desta forma, pode-se aproveitar o tempo ocioso de computadores pessoais, dispositivos móveis e consoles de *videogames* para acelerar o processamento dessas aplicações (ANDERSON, 2004).

Projetos de computação pública existem desde meados dos anos 1990 e consistem em um computador que atua como um servidor, enviando tarefas para que estas sejam processadas por computadores voluntários em seu tempo ocioso. Ao fim do processamento, estes computadores voluntários enviam seus resultados ao servidor, para que sejam analisados (ANDERSON, 2004).

A computação pública se assemelha aos *grids* computacionais uma vez que ambos objetivam uma melhor utilização dos recursos computacionais existentes. No entanto, há algumas diferenças significativas entre ambos os paradigmas. Os *grids* computacionais em geral envolvem recursos computacionais (supercomputadores, *clusters* e computadores) de universidades, laboratórios de pesquisa e empresas. Estes recursos caracterizam-se por serem gerenciados por profissionais de TI, permanecerem ligados durante a maior parte do tempo e serem conectados à internet por *links* dedicados e de alta largura de banda. Além disso, comportamentos maliciosos, como a falsificação intencional de resultados, são atípicos (ANDERSON, 2004).

Em contraste, a computação pública envolve pequenos grupos acadêmicos com poder computacional e mão-de-obra limitados, sendo que os recursos computacionais são cedidos por pessoas que possuem computadores com sistemas operacionais Windows, Linux ou MacOS, conectados à internet através de conexões do tipo *cable modem* ou DSL (*Digital Subscriber Line*). Normalmente, pela natureza das conexões à internet, estas máquinas encontram-se por trás de tradutores de endereço de rede (*network-address translators*, ou NAT) ou ainda protegidos por *firewalls*. Além disso, os computadores são frequentemente desligados ou desconectados da internet. Os participantes de projetos de computação pública podem ser pessoas sem experiência com computadores, e participam de projetos somente quando têm interesse e recebem "incentivos", como protetores de tela e créditos no projeto. Por fim, os projetos não possuem controle sobre os participantes, nem meios de garantir que não haverá fraudes (ANDERSON, 2004).

Devido a estas diferenças, os *middlewares* de projetos de computação pública e de grids computacionais são especializados para seu paradigma de processamento. De fato, os *middlewares* para computação pública precisam possuir mecanismos de segurança que diminuam a chance do usuário fraudar resultados e precisam ser resilientes à perda de conexão ou quedas de energia, para evitar que o processamento realizado seja perdido.

3.1 Plataforma BOINC

O BOINC (*Berkeley Open Infrastructure for Network Computing*) é uma plataforma para computação pública, que foi desenvolvida pelo Laboratório de Ciências Espaciais da Universidade da Califórnia em Berkeley. O grupo que trabalha no desenvolvimento do BOINC é o mesmo que desenvolveu e continua operando o projeto SETI@Home (BOINC Documentation, 2015).

O objetivo inicial com o BOINC era facilitar o desenvolvimento de aplicações que necessitavam de uma grande capacidade de processamento, de modo que estas pudessem utilizar-se do paradigma de computação pública. O BOINC é capaz de controlar todas as necessidades da computação pública e facilitar aos desenvolvedores a adaptação de aplicações já existentes para este paradigma. O BOINC é um software de código aberto e encontra-se disponível no endereço `http://boinc.berkeley.edu` (ANDERSON, 2004; SANTOS, 2005).

O BOINC pode ser dividido em três componentes básicos: servidor, cliente e aplicação. Os projetos BOINC são identificados por uma URL-mestre, que é a página principal do *website* e que é utilizada como diretório dos servidores de agendamento. Os voluntários cadastram-se nos projetos, que podem envolver uma ou mais aplicações e cujo conjunto de aplicações pode ser alterado ao longo do tempo.

O processamento a ser realizado é dividido em pequenas tarefas, chamadas de *workunits* (ou WU), sendo que cada tarefa é enviada pelo servidor a um ou mais clientes para ser processada. O resultado então é retornado para o servidor para uma análise (ANDERSON, 2004; SMIDERLE, 2012). A Figura 3.1 ilustra a arquitetura da plataforma BOINC.

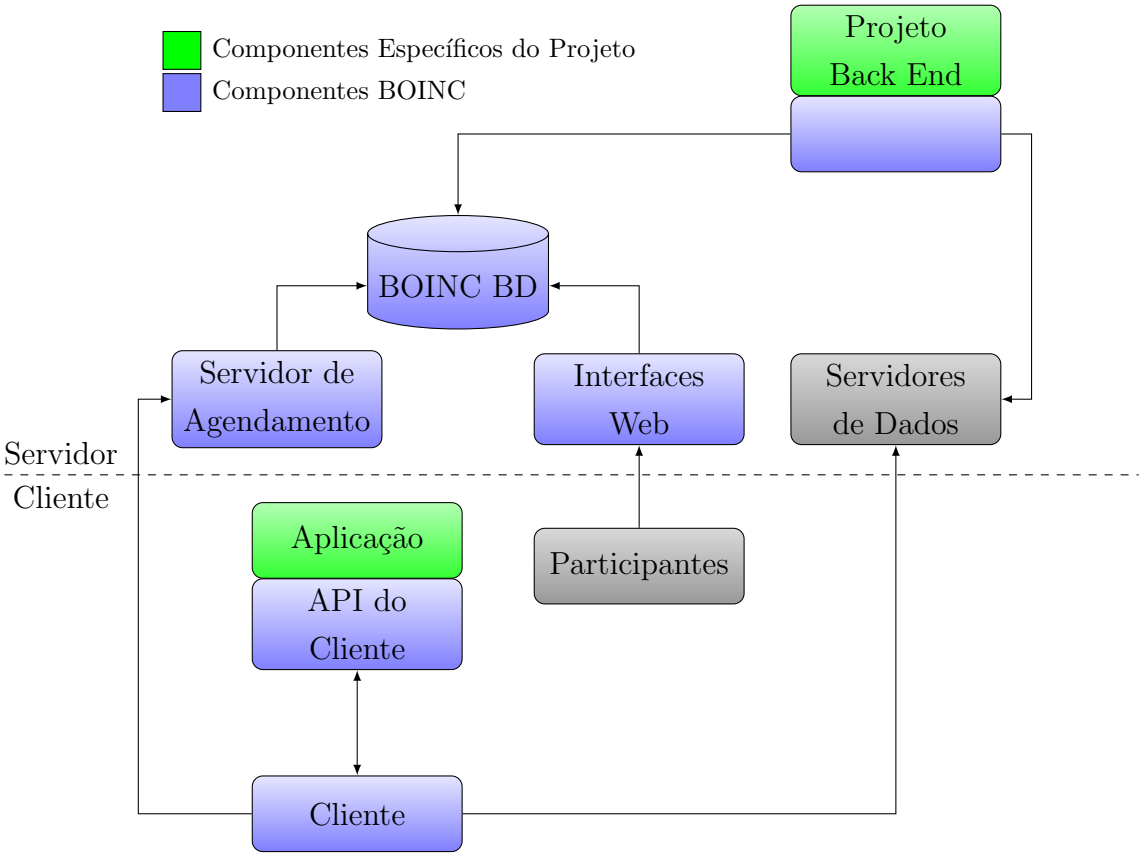


Figura 3.1: Arquitetura BOINC (SMIDERLE, 2012).

A plataforma BOINC possui um sistema de controle de pontuação dos usuários, que é utilizada como forma de recompensa para os usuários que participam ativamente do projeto. A fim de evitar fraudes no sistema de pontuação, cada WU pode ser enviada a mais de um cliente e, ao receber o retorno de cada um destes processamentos, o servidor compara os resultados obtidos com o intuito de verificar se os resultados retornados não foram fraudados. Os pontos somente são creditados aos usuários após esta verificação (BOINC Documentation, 2015; SMIDERLE, 2012).

A pontuação solicitada por cada cliente é baseada no tempo de processamento fornecido. A quantidade de pontos que os usuários receberão é a mínima entre as pontuações solicitadas por todos os clientes que processaram a mesma WU (BOINC Documentation, 2015; SMIDERLE, 2012). Como pode ser observado na Figura 3.2, o cliente 1 solicitou 123 pontos ao servidor após processar uma WU. Já o cliente 2, após processar a mesma WU, solicitou 125 pontos. O servidor irá, então, validar os resultados enviados pelos clientes, e caso estes sejam idênticos, creditará 123 pontos

a ambos os clientes.

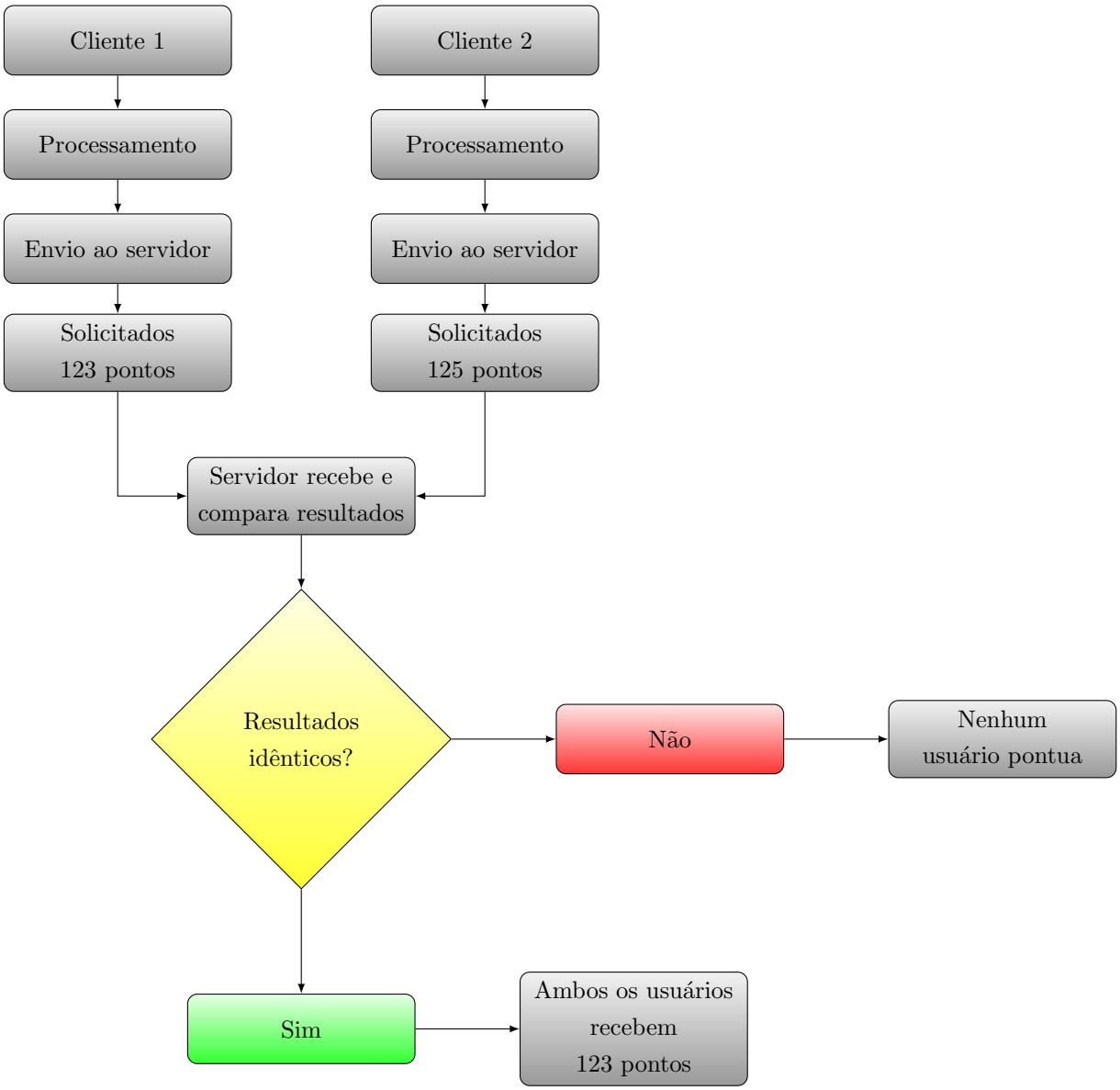


Figura 3.2: Funcionamento do sistema de recompensas do BOINC (BOINC Documentation, 2015).

3.1.1 Servidor BOINC

O servidor de um projeto BOINC é centralizado em torno de um banco de dados relacional, onde são armazenadas as informações de cada projeto. Estas informações incluem a lista de usuários participantes, a relação das WUs disponíveis, enviadas e processadas, além do estado de cada uma delas. Além disso, há um servidor *Web* para a interação com os participantes e a distribuição das WUs; um servidor de dados para armazenamento das WUs; e um servidor de agendamento com o qual o cliente comunica-se para solicitar novas WUs (ANDERSON, 2004; SANTOS, 2005; SMIDERLE, 2012). A Figura 3.3 ilustra a interação entre o cliente e o servidor BOINC.

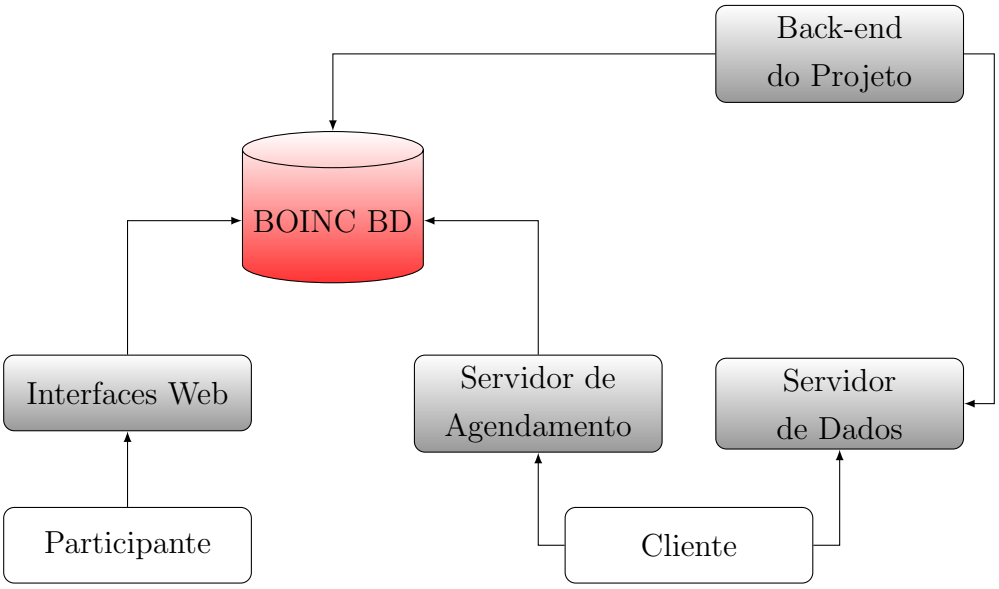


Figura 3.3: Servidor BOINC e sua interação com o cliente (SMIDERLE, 2012).

A computação pública está sujeita a falhas, principalmente relacionadas a computadores com problemas e participantes mal-intencionados. Por isso, o BOINC oferece suporte ao processamento redundante, que é um mecanismo para identificação e rejeição de resultados errôneos. Os administradores dos projetos podem indicar um número N de resultados a serem criados para cada WU. Depois que o número mínimo de WUs forem distribuídas e concluídas, uma função específica é invocada para comparar os resultados e selecionar um resultado canônico. Se o consenso não for obtido, ou os resultados falharem, o BOINC repete o processo de distribuição e obtenção de resultados, até que o resultado desejado seja obtido, ou sejam esgotados um número máximo de tentativas (ANDERSON, 2004).

Algumas aplicações de processamento numérico podem apresentar resultados diferentes para uma mesma WU, dependendo da arquitetura, sistema operacional, compilador e *flags* de compilação. Nestes casos, pode tornar-se difícil distinguir resultados errados de resultados que estão corretos, mas que apresentam diferenças devido a estes fatores. Para estes casos, o BOINC oferece uma opção de redundância homogênea, que garante que o agendador enviará uma WU somente para máquinas com o mesmo sistema operacional e com processadores do mesmo fabricante (ANDERSON, 2004).

Para implementar o processamento redundante, o BOINC utiliza os seguintes processos: o **transacionador**, responsável por controlar o estado transacional de cada WU; o **validador**, responsável por verificar os resultados enviados pelos clientes e garantir que os mesmos sejam válidos de acordo com os critérios determinados pelo administrador do projeto; o **assimilador**, responsável por realizar tarefas de pós-processamento sobre os resultados enviados pelos clientes; e o ***file deleter***, responsável por eliminar as WUs já assimiladas. O validador e o assimilador fazem

parte do *back-end* do projeto, ou seja, realizam o processamento específico de cada projeto. Desta forma, estes dois programas devem ser implementados pelo administrador do sistema, de modo a suprir as necessidades individuais de cada projeto (SMIDERLE, 2012).

O validador é responsável por comparar os resultados retornados pelos clientes. Os resultados inválidos podem ser fruto de fraudes, de erros no cliente ou de falhas na aplicação. Cada vez que um resultado de uma WU é recebido, o validador seta o estado da WU como "precisa ser validada". Em seguida, o validador compara todos os resultados desta WU de modo a garantir um resultado canônico (SMIDERLE, 2012).

O assimilador é o responsável por realizar as operações necessárias sobre os resultados das WUs. Como exemplo de operações, pode-se citar o pós-processamento sobre o resultado obtido ou o armazenamento destas informações. O BOINC oferece dois modelos de assimilador, sendo um em linguagem de programação C e outro em linguagem Python. Ambos possuem uma função `assimilate_handler` que deve ser definida pelo administrador do projeto. Nesta função deve-se implementar as operações que se deseja realizar sobre cada resultado parcial. Após a passagem da WU pelo assimilador, seu estado é modificado para "finalizado"e a WU fica disponível para ser descartada pelo *file deleter* (SMIDERLE, 2012).

3.1.2 Cliente BOINC

O cliente BOINC é o aplicativo executado no computador do voluntário. Este cliente é comum a todos os projetos, sendo mantido pela equipe desenvolvedora do BOINC. Este cliente geralmente é disponibilizado nas páginas dos projetos de computação pública que utilizam a plataforma BOINC, mas também pode ser obtido através do *website* do BOINC (`http://boinc.berkeley.edu/`) (SMIDERLE, 2012).

O cliente BOINC é responsável pela autenticação dos usuários e por realizar o *download* das aplicações específicas dos projetos e das WUs, além de oferecer a interface com o usuário e realizar o controle de utilização dos recursos do computador voluntário, permitindo ao usuário a seleção de opções, como o percentual de processamento, memória e armazenamento em disco que serão ocupados pelos projetos de computação pública gerenciados pelo cliente. Pode-se ainda selecionar o percentual de tempo destinado a cada projeto, visto que um cliente pode gerenciar múltiplos projetos simultaneamente (SMIDERLE, 2012). Na Figura 3.4 há uma ilustração da tela de configuração das opções do cliente BOINC.

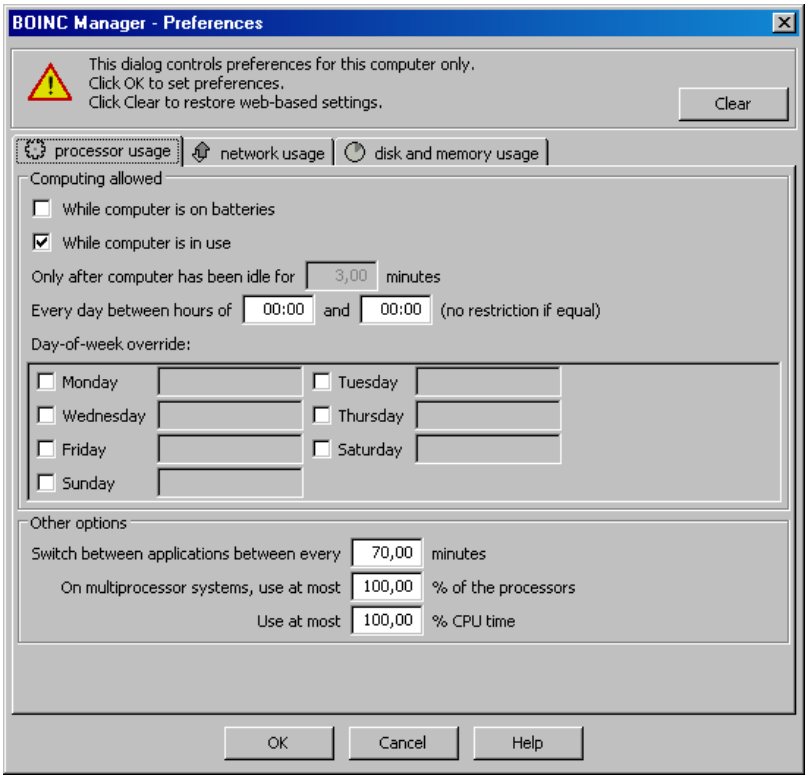


Figura 3.4: Configuração das opções do cliente BOINC.

O BOINC oferece clientes para vários sistemas operacionais e arquiteturas, sendo que os sistemas operacionais Windows, Linux e MacOS são suportados, assim como o Android. Atualmente, apesar do MacOS ser suportado, o cliente oficial do BOINC não possui suporte para o iOS por razões técnicas e legais que impõem barreiras na implementação de um cliente para este sistema operacional (BOINC Documentation, 2015). Na Figura 3.5 tem-se uma ilustração mostrando a estrutura do cliente BOINC.

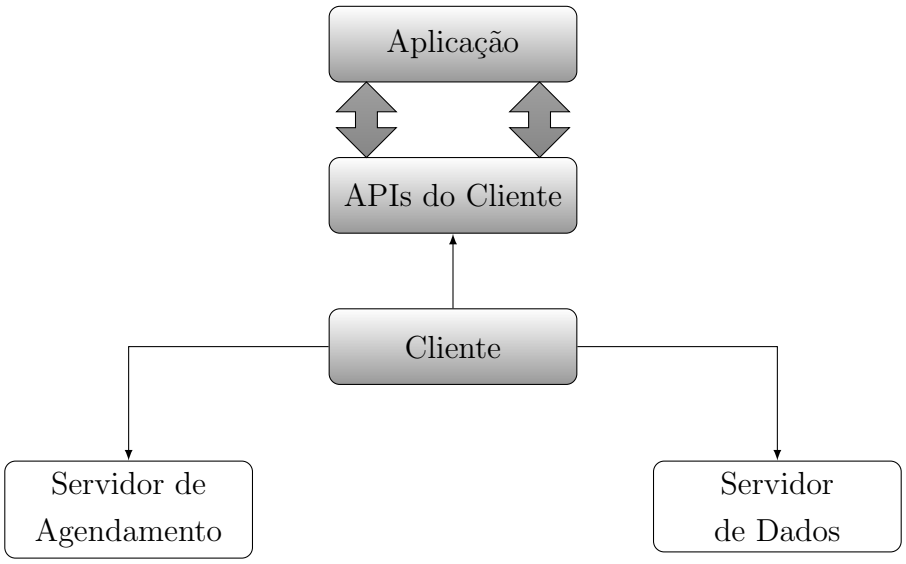


Figura 3.5: Arquitetura do cliente BOINC (SMIDERLE, 2012).

3.1.3 Aplicação

A aplicação é que efetivamente executa as computações necessárias em projetos de computação pública gerenciados pelo BOINC. Para cada plataforma suportada, uma aplicação diferente deve ser implementada, respeitando as limitações individuais de cada plataforma. Assim, o administrador do projeto deve se responsabilizar por oferecer esta aplicação com suporte a diferentes plataformas, para que possa ser executada por uma gama maior de usuários (SANTOS, 2005).

A aplicação não acessa funções do sistema operacional do computador cliente e não tem acesso ao *hardware* diretamente. Para realizar este tipo de operação, a aplicação deve utilizar a API do BOINC. De fato, a API (*Application Programming Interface*) fornece o isolamento necessário entre a aplicação e o computador do usuário, a fim de reduzir o risco de aplicações mal-intencionadas utilizarem o BOINC para roubar informações. Esta API oferece funções para o gerenciamento de arquivos, de armazenamento de estado de processamento e de seções críticas, além das funções básicas necessárias para a inicialização e a finalização de processamentos utilizando o BOINC (BOINC Documentation, 2015).

Além disso, a API BOINC oferece a possibilidade da aplicação renderizar gráficos no computador do usuário, oferecendo uma proteção de tela animada durante o tempo de processamento. A utilização destas funções torna o projeto mais atrativo para o usuário. A aplicação poderia, por exemplo, utilizar este recurso para exibir o progresso do processamento das WUs e do projeto como um todo (SMIDERLE, 2012).

A API gráfica do BOINC oferece suporte multiplataforma para o desenvolvimento de aplicações gráficas. Esta API tem como base a biblioteca OpenGL e possui funções próprias para a inicialização e atualização dos gráficos, além de possuir funções para a detecção de movimentos do mouse e utilização de teclado. Uma descrição mais completa da API gráfica do BOINC encontra-se na documentação do projeto (BOINC Documentation, 2015).

4 ESTUDO DE CASO

A fim desenvolver aplicações que possam ser executadas na plataforma BOINC, é necessário instalar as bibliotecas e os aplicativos do BOINC no computador que será utilizado para o desenvolvimento. Algumas distribuições de Linux oferecem pacotes com as aplicações e configurações necessárias, porém é possível obter o código-fonte do projeto e compilá-lo diretamente no computador que será utilizado.

No computador que atuará como servidor BOINC deverá ser instalado o Apache HTTP Server (The Apache Foundation, 2015) e um servidor de banco de dados MySQL (MYSQL, 2015), além do próprio software do BOINC (BOINC Documentation, 2015). Neste servidor, deverá ser criado um projeto que conterá todas as aplicações disponíveis para as diferentes plataformas suportadas. Por exemplo, se o projeto suportar as plataformas Windows 32 e 64-bit, Linux 32 e 64-bit e Android ARM, estas cinco aplicações deverão estar contidas no servidor, dentro da diretório do projeto a que se referem, que indica a aplicação, a versão e a plataforma (sistema operacional e arquitetura). A estrutura de diretórios das aplicações disponíveis para diferentes plataformas está ilustrada na Figura 4.1.

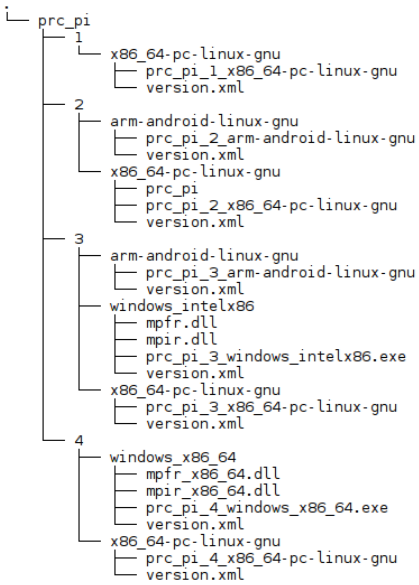


Figura 4.1: Estrutura de pastas das aplicações do projeto BOINC.

Como pode ser observado na Figura 4.1, cada aplicação contém em seu diretório um arquivo XML chamado `version.xml`, que contém as informações necessárias dos executáveis que devem ser baixados pelo cliente BOINC a fim de iniciar o processamento. Um exemplo de arquivo `version.xml` encontra-se descrito no Quadro 4.1. Neste arquivo, a *tag* `<file>` descreve os arquivos presentes na versão. A *tag* `<physical_name>` indica o nome físico do arquivo no diretório da versão. A *tag* `<main_program/>` indica se o arquivo em questão se refere ao programa principal da versão. A *tag* `<copy_file/>` indica se o arquivo deverá ser copiado para o diretório de execução do cliente ou apenas deverá possuir um vínculo simbólico. Por fim, a *tag* `<logical_name>` indica o nome que o programa assumirá no diretório de execução do cliente.

Quadro 4.1: Arquivo `version.xml`.

```
1 <version>
2         <file>
3                 <physical_name>prc_pi_4_x86_64-pc-linux-gnu</physical_name>
4                 <main_program />
5                 <copy_file />
6                 <logical_name>prc_pi</logical_name>
7         </file>
8 </version>
```

O BOINC oferece uma ferramenta para a configuração dos projetos, que é distribuída juntamente com o código-fonte, chamada `make_project`. Esta ferramenta, entre outras funcionalidades, gera os diretórios e arquivos necessários; cria o banco de dados e inicializa os *daemons* do BOINC referentes ao projeto. Após a criação do projeto, é necessário desenvolver as partes específicas do projeto, que são o assimilador e o validador, quando necessários, e a aplicação propriamente dita.

4.1 Servidor BOINC

Após a instalação do *software* necessário no servidor, é preciso criar o projeto e efetuar as configurações específicas. Esta seção descreve sucitamente os procedimentos que foram realizados para esta configuração.

4.1.1 Configuração do servidor

O computador utilizado como servidor neste projeto, que também serviu como ambiente de desenvolvimento e como cliente durante a fase de testes, foi um computador com processador Intel Core i7-3630QM @ 2.40 GHz, com 8 GB de memória RAM e disco rígido de 120 GB. Este computador conta com sistema operacional Fedora 21 (versão do *kernel* Linux 4.0.5). O software e as bibliotecas BOINC necessárias ao desenvolvimento e aos testes foram compiladas a partir do código-fonte e instaladas para que pudessem ser utilizadas.

Além dos softwares específicos do BOINC, foram instalados o servidor de banco de dados MySQL, versão 15.1, e o Apache HTTP Server, versão 2.4.12. Estes dois softwares foram instalados diretamente dos repositórios de software que são oferecidos pelo sistema operacional Fedora. Por fim, o projeto foi gerado utilizando o script `make_project`, que é fornecido pela plataforma BOINC.

4.1.2 Criação das WUs

O processo de criação das WUs é uma parte fundamental do processamento das aplicações BOINC. Esta seção descreve como foram geradas as WUs para o cálculo dos dígitos do Pi.

4.1.2.1 Templates das WUs

Foram gerados dois arquivos de *templates*, que são necessários à geração e processamento das WUs. Estes arquivos residem no diretório *templates*, presente no diretório-raiz do projeto BOINC. O *template* que descreve os dados de entrada das WUs foi nomeado `input` e configurado conforme descrito no Quadro 4.2.

Quadro 4.2: *Template* de entrada `input.xml`.

```
<input_template>
        <file_info>
                <number>0</number>
        </file_info>
        <workunit>
                <file_ref>
                        <file_number>0</file_number>
                        <open_name>infile</open_name>
                        <copy_file />
                </file_ref>

                <rsc_fpops_est>1e9</rsc_fpops_est>
                <rsc_fpops_bound>1e15</rsc_fpops_bound>
                <rsc_memory_bound>262144000</rsc_memory_bound>
                <rsc_disk_bound>52428800</rsc_disk_bound>
                <delay_bound>86400</delay_bound>
                <min_quorum>80</min_quorum>
                <target_nresults>120</target_nresults>
                <max_error_results>40</max_error_results>
                <max_total_results>120</max_total_results>
                <max_success_results>120</max_success_results>
        </workunit>
</input_template>
```

No *template* de entrada, há uma série de *tags* que podem ser especificadas em um arquivo XML ou fornecidas dinamicamente no momento da geração da WU (BOINC Documentation, 2015). Neste projeto, optou-se por parametrizar estas informações de modo estático, em um arquivo XML.

A *tag* `<file_info>`, existente na raiz do arquivo XML, descreve o arquivo de entrada da aplicação. A *tag* `<number>` fornece uma numeração para os arquivos

de entrada, que será utilizado pelo BOINC para gerar os nomes físicos dos arquivos para envio aos clientes (BOINC Documentation, 2015).

Na *tag* `<workunit>`, são informados parâmetros que dizem respeito às WUs. A *tag* `<file_ref>` descreve as informações lógicas dos arquivos. A *tag* `<file_number>` é o número do arquivo de entrada e a *tag* `<open_name>` é o nome lógico do arquivo, ou seja, o nome que a aplicação utilizará para buscar o nome físico do arquivo através da função `boinc_resolve_filename` (ver Seção 4.2). A *tag* `<copy_file />` indica que o arquivo será copiado para o diretório corrente onde será executada a WU no computador voluntário (BOINC Documentation, 2015).

A seguir, há *tags* que informam ao servidor e aos clientes a natureza da WU que será processada. A *tag* `<rsc_fpops_est>` informa uma estimativa de quantidade de operações de ponto-flutuante que serão realizadas durante o processamento. Esta estimativa foi baseada no *benchmark* realizado pelo BOINC no computador servidor, multiplicado pelo tempo que o computador demorou para processar uma das WUs. A *tag* `<rsc_fpops_bound>` é o limite superior de operações de ponto-flutuante que uma WU irá necessitar para ser procesada. A *tag* `<rsc_memory_bound>` é a quantidade máxima de memória, em bytes, necessária para o processamento de uma WU. A *tag* `<rsc_disk_bound>` indica o espaço máximo em disco, em bytes, que será utilizado pela WU para o seu processamento, incluindo arquivos intermediários. Por fim, a *tag* `<delay_bound>` é o limite de tempo, em segundos, que a WU poderá permanecer em processamento no cliente antes de ser abortada. Os demais parâmetros, descritos pelas *tags* `<min_quorum>`, `<target_nresults>`, `<max_error_results>`, `<max_total_results>` e `<max_success_results>` dizem respeito à quantidade de resultados que devem ser criados e quantos resultados válidos e inválidos serão tolerados antes que a WU seja declarada concluída ou abortada por erros (BOINC Documentation, 2015).

Quadro 4.3: *Template* de saída `output.xml`.

```
1  <output_template>
2          <file_info>
3                  <name><OUTFILE_0/></name>
4                  <generated_locally />
5                  <upload_when_present/>
6                  <max_nbytes>32768</max_nbytes>
7                  <url><UPLOAD_URL/></url>
8          </file_info>
9
10         <result>
11                 <file_ref>
12                         <file_name><OUTFILE_0/></file_name>
13                         <open_name>outfile</open_name>
14                 </file_ref>
15         </result>
16 </output_template>
```

O *template* que descreve os dados de saída foi nomeado `output` e foi configurado conforme o Quadro 4.3. Neste *template* há uma quantidade menor de informações, em relação ao *template* de entrada e a primeira *tag*, `<file_info>`, contém informações sobre os arquivos de saída das WUs, sendo composto pelas *tags* `<name>`, `<generated_locally/>`, `<upload_when_present/>`, `<max_nbytes>` e `<url>`. A *tag* `<name>` contém o nome físico do arquivo de saída, que no caso dos resultados, deve ser preenchido com a *tag* `<OUTFILE_N>`, em que N é onúmero do arquivo. O BOINC utiliza esta tag para gerar um nome físico baseado no nome da WU. Por exemplo, para uma *workunit* de nome `wu_test_0`, o arquivo de saída poderia se chamar `wu_test_0_0`. A *tag* `<generated_locally>` é necessária para arquivos de saída e indica que o arquivo será gerado localmente, ou seja, no diretório de execução do cliente BOINC. A *tag* `<upload_when_present>` é desnecessária para clientes BOINC após a versão 7.0, mas deve ser mantida por questões de compatibilidade com clientes antigos. A *tag* `<max_nbytes>` indica o tamanho máximo do arquivo de saída, para que este seja enviado ao servidor sem que a WU seja marcada com erro. Por fim, a *tag* `<url>` contém a URL que deve ser utilizada pelo cliente para envio dos arquivos de saída. Na *tag* `<url>` pode-se informar um endereço fixo ou a *tag* `<UPLOAD_URL/>`, que busca a informação do arquivo de configuração do projeto (BOINC Documentation, 2015). A estrutura da *tag* `<file_info>` pode ser visualizada a partir da linha 2 do Quadro 4.3.

A segunda *tag*, `<result>`, contém as informações a respeito do arquivo de saída que deverá ser gerado, sendo composta pelas *tags* `<file_name>` e `<open_name>`. Assim como no *template* de entrada, no *template* de saída a *tag* `<file_ref>` contém as informações de nome lógico e nome físico do arquivo, sendo que a *tag* `<file_name>` indica o nome físico que o arquivo assumirá e a *tag* `<open_name>` é o nome lógico do arquivo que será utilizado pela aplicação (BOINC Documentation, 2015).

Mais informações a respeito dos *templates* de entrada e saída das *workunits* do BOINC podem ser encontrados na documentação oficial da plataforma, disponível em `https://boinc.berkeley.edu/trac/wiki/ProjectMain`.

4.1.2.2 Gerador de WUs

Neste projeto, foi desenvolvido um programa gerador de *workunits*, cujo código encontra-se na Seção 7.3 do Anexo A. Este programa utiliza a função `create_work`, que é fornecida pela API do BOINC, para a geração das WUs. Esta função recebe informações do *template* de entrada, da conexão com o banco de dados e do *template* de saída. O programa desenvolvido recebe dois parâmetros de linha de comando para a geração de *workunits*, que são o número do primeiro dígito e o número do último dígito do Pi que se deseja calcular. Após, a função `read_file_malloc`

é executada para ler o *template* de entrada. Com os dados do *template* de entrada e as informações dos dígitos a calcular, o programa cria um arquivo de entrada no diretório de *downloads* do BOINC, contendo o dígito a calcular, cujo nome tem a forma *wu_inputDDD*, em que DDD é o dígito que será calculado naquela *workunit*.

O programa então invoca a função `create_work`, com as informações da *workunit*, o *template* de entrada carregado, o caminho do *template* de saída, especificação e quantidade dos arquivos de entrada e a configuração do banco de dados. Por fim, a função armazena as *workunits* geradas no banco de dados, sendo que o servidor já pode enviá-las para o processamento nos clientes.

4.1.2.3 Controle das WUs

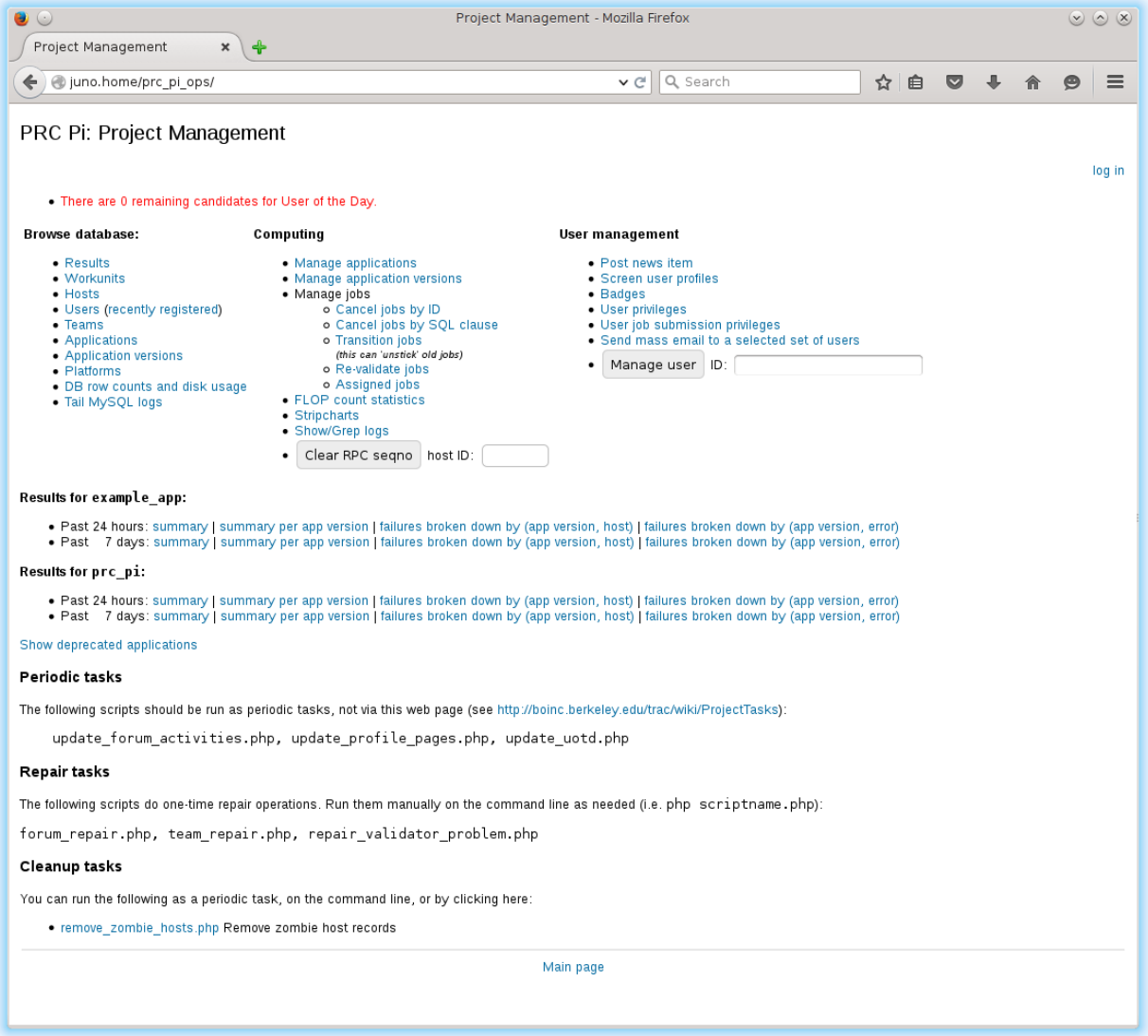


Figura 4.2: Tela de gerenciamento do servidor BOINC.

O controle das *workunits*, dos resultados, dos clientes e de outras informações pode ser feito através da interface *web* gerada pelo projeto BOINC. Além das funcionalidades já citadas, a interface ainda permite a moderação dos fóruns, que também são gerados pelo projeto, e o controle dos usuários, entre outras funções. Uma das telas da interface *web* pode ser visualizada na Figura 4.2.

4.1.2.4 Validador e assimilador

Neste projeto, foi desenvolvido um assimilador de resultados, que armazena em um arquivo-texto informações das *workunits* que foram finalizadas com sucesso. As informações armazenadas pelo assimilador são o número da WU, o resultado do cálculo do dígito de Pi em base hexadecimal e o valor equivalente em base decimal. No Quadro 4.4 há temos um exemplo do conteúdo do arquivo que foi gerado pelo assimilador.

Quadro 4.4: Exemplo do arquivo de saída do assimilador.

```
3023,"9F28FE6ED9",0.621719266951004101429134607315063476562 5
4999,"1CAD181156",0.112016205055624595843255519866943359375 0
5012,"95E0333E92",0.585452273155169677920639514923095703125 0
7967,"393A835315",0.223549087318133388180285692214965820312 5
```

Para a utilização de um assimilador específico, ou seja, um assimilador diferente daquele que é disponibilizado pela plataforma BOINC, foi necessária a alteração do arquivo `config.xml` a fim de adicionar o novo *daemon*, conforme descrito no Quadro 4.5.

Quadro 4.5: Configuração do *daemon* do assimilador no arquivo `config.xml`.

```
<daemon>
        <cmd>pi_assimilator -d 3 --app prc_pi</cmd>
</daemon>
```

Em função do resultado da computação ser uma string hexadecimal, não foi desenvolvido um validador de resultados, tendo sido utilizado para esta finalidade o validador bit-a-bit que é fornecido com a plataforma BOINC.

4.2 Aplicação

A aplicação desenvolvida neste trabalho implementou o algoritmo BBP para cálculo dos dígitos hexadecimais do Pi. Este algoritmo foi descrito na Seção 2.5.1 e foi selecionado para que fosse possível distribuir as tarefas sem a necessidade de calcular os dígitos predecessores antes.

O código-fonte da aplicação foi escrito em linguagem de programação C, utilizando a API do BOINC para o gerenciamento de arquivos. Para os cálculos de ponto

flutuante, foi utilizada a biblioteca GNU MPFR (*GNU Multiple Precision Floating-Point Reliable Library*), que é um aprimoramento da biblioteca GMP (*GNU Mathematical Precision*). A utilização desta biblioteca se deve à necessidade de precisão arbitrária e métodos de arredondamento confiáveis para obter valores precisos nos cálculos em ponto-flutuante.

O funcionamento da aplicação é simples: o cliente BOINC faz o *download* de um arquivo de entrada, especificado no *template* de entrada, que contém a posição do dígito de Pi que deve ser calculado. Um exemplo do arquivo de entrada encontra-se no Quadro 4.6.

Quadro 4.6: Exemplo do arquivo de entrada para o cálculo do dígito 5000.

```
1 5000
```

A leitura do arquivo de entrada passa por duas etapas, sendo que a primeira consiste na resolução do nome do arquivo. Os arquivos no BOINC possuem um nome lógico e um nome físico. O nome lógico é aquele especificado no *template* de entrada (neste caso, `infile`) e é usado pela aplicação para resolver o nome físico, que é o nome que o arquivo realmente possui no sistema de arquivos do computador cliente. Esta etapa é realizada pela função `boinc_resolve_filename` (linha 7 do Quadro 4.7). A segunda etapa é a abertura do arquivo através da API BOINC. Esta etapa utiliza a função `boinc_fopen`, que acessa o sistema de arquivos e retorna para a aplicação um descritor para o arquivo de entrada que será manipulado pela aplicação (linha 16 do Quadro 4.7).

Quadro 4.7: Função para a leitura do arquivo que contém a posição do dígito do Pi que será calculado.

```
1  static long read_digit_position() {
2          char filename[filename_length];
3          int retval;
4          long position;
5          FILE *input;
6
7          retval = boinc_resolve_filename("infile",
8                          filename,
9                          filename_length);
10
11         if(retval) {
12                 perror("couldn't open file");
13                 exit(-1);
14         }
15
16         input = boinc_fopen(filename, "r");
17         fscanf(input, "%ld", &position);
18         fclose(input);
19
20         return position;
21 }
```

Após a leitura, a aplicação calcula o valor do dígito especificado no arquivo de entrada e executa o método BBP para obter um valor em ponto-flutuante que representa o dígito do Pi naquela posição. Posteriormente, a aplicação converte o valor calculado para a base hexadecimal, e armazena este valor em um arquivo de saída, que será enviado ao servidor. O código-fonte desta aplicação encontra-se na Seção 7.1 do Anexo A.

4.2.1 Compilação da aplicação

A aplicação foi compilada para quatro plataformas distintas: Linux 64-bit, Windows 32-bit/64-bit e Android ARM. Não houve a necessidade de alterações no código-fonte para a geração destas quatro versões. Embora a compilação para estas quatro plataformas tenha sido possível, a versão para Windows 32-bit não foi testada devido à indisponibilidade de computadores com este sistema operacional.

A compilação para Linux da aplicação apresenta dependências para a biblioteca GNU MPFR, que deverá ser instalada nos computadores que farão o processamento antes que o computador seja vinculado ao projeto no BOINC. Em função do funcionamento das bibliotecas dinâmicas no Linux, optou-se por não distribuir as bibliotecas juntamente com a aplicação BOINC. Como forma de automatização da compilação para Linux, foi utilizado o arquivo *Makefile*, que encontra-se na Seção 7.1.1 do Anexo A.

Para a compilação no sistema operacional Windows, foi utilizado o processo de compilação que foi disponibilizado por Pavel Holoborodko e que encontra-se disponível em `http://www.holoborodko.com/pavel/mpfr/`. As bibliotecas disponíveis nesta página foram vinculadas à versão da aplicação para Windows, de modo que o BOINC efetue o *download* da aplicação e das bibliotecas necessárias antes de iniciar a execução, não sendo necessária intervenção do usuário para esta finalidade.

No sistema operacional Android, a aplicação foi compilada com vínculos estáticos à biblioteca GNU MPFR, de maneira que o arquivo principal da aplicação apresenta um tamanho maior (em bytes), porém sem dependências que precisem ser instaladas. Esta decisão se justificou devido ao fato da instalação de pacotes de *software* no Android não ser tão trivial. Desse modo, a compilação com vínculos estáticos se mostrou a opção mais simples e prática para a disponibilização da aplicação a usuários com pouco conhecimento em informática. Para que esta compilação se tornasse possível, foi necessária a compilação das bibliotecas GMP, versão 6.0.0, e MPFR, versão 3.1.2, para a plataforma Android ARM. Para automatizar a compilação para Android, foi utilizado o script `build_build_app_arm.sh`, fornecido pelo BOINC, em conjunto com o arquivo *Makefile*, que encontra-se descrito na Seção 7.1.2 do Anexo A.

5 RESULTADOS OBTIDOS

Após a conclusão da implementação do projeto, foram realizados testes com o objetivo de validar o programa desenvolvido. Por fim, foram realizadas medições do tempo de execução nas três plataformas. Para estes testes, foram utilizados quatro dispositivos distintos. Os resultados dos testes e das medições de tempo estão descritas neste capítulo.

5.1 Validação da aplicação

O algoritmo e a aplicação desenvolvida foram validados através da execução de 10000 *workunits* para calcular todos os dígitos do Pi do índice 0 até o índice 9999. O assimilador desenvolvido armazenou os resultados em um arquivo-texto separado por vírgulas. As informações armazenadas foram a posição do dígito, o valor calculado em base hexadecimal e valor calculado em base decimal. A validação dos resultados obtidos foi realizada de modo automático por uma aplicação, que foi desenvolvida em linguagem Java. Essa aplicação leu o arquivo de resultados e comparou os valores com os resultados retornados pelo *webservice* do projeto HexPi (HexPi, 2015). A execução do validador automatizado confirmou que todos os valores obtidos pela aplicação eram válidos. Alguns dos valores hexadecimais do Pi calculados pela aplicação estão listados na Tabela 5.1. As WUs utilizadas na validação foram executadas nas três plataformas, embora nem todas tenham tido tarefas processadas por todas as plataformas em função do escalonamento feito pelo servidor BOINC. Os resultados que foram processados pelas três plataformas resultaram em valores idênticos.

Tabela 5.1: Listagem de alguns valores hexadecimais do Pi calculados pela aplicação.

Índice do dígito	**Valor hexadecimal**
0	243F6A8885
10	A308D31319
50	2EFA98EC4E
100	29B7C97C50
500	86AF7C72E9
1000	49F1C09B07
5000	CAD181156B
9999	68AC8FCFB8

5.2 Teste de desempenho

Para a verificação do desempenho nas três plataformas distintas, foi gerada uma *workunit* para o cálculo do dígito de número 100.000 do Pi. A escolha do dígito 100.000 deve-se ao fato deste necessitar um número maior de iterações para ser calculado. Esta WU foi executada nos dispositivos que estão descritos na Tabela 5.2.

Tabela 5.2: Computadores utilizados para tabular o desempenho do aplicativo com configuração e sistema operacional.

Computador nº	**Sistema Operacional**	**Processador**	**Memória RAM**
1	Linux 4.0.5 64-bit (Fedora 21)	Core i5-M460 @ 2.53 GHz	4 GB
2	Windows 8.1 64-bit	Core i5-M460 @ 2.53 GHz	4 GB
3	Android 4.0.4	ARM Cortex-A9 @ 1.2 GHz	1 GB
4	Android 4.3	Qualcomm Snapdragon S4 @ 1.73 GHz	1 GB

O comparativo dos tempos de execução da *workunit* gerada para a medição do desempenho em cada um dos dispositivos está descrito na Tabela 5.3. Cada um dos computadores processou 35 vezes a mesma WU, sendo que na tabela estão apresentadas as médias (em segundos) e desvio-padrão dos tempos de processamento para cada uma das arquiteturas.

Tabela 5.3: Resultados do processamento da *workunit* para o cálculo do dígito 100.000 do Pi.

Computador	**Sistema Operacional**	**Média**	**Desvio-padrão**
1	Linux 4.0.5	115.008294	1.1281476286
2	Windows 8.1	748.115957	5.3076358845
3	Android 4.0.4	450.880645	0.8159258807
4	Android 4.3	283.675691	2.5760515465

É possível observar a partir dos resultados que o pior desempenho ficou com o sistema operacional Windows, enquanto o sistema operacional Linux teve o melhor de todos os desempenhos. É importante também destacar que ambas as execuções, apesar de terem ocorrido em dois sistemas operacionais distintos, foram realizadas em um mesmo computador. Os dispositivos com Android, apesar de seu baixo poder de processamento quando comparados com os outros dois dispositivos, tiveram desempenho melhor que o Windows. Esta observação a respeito do mau desempenho do Windows em aplicações de computação pública com grande volume de cálculos já havia sido feita por SMIDERLE (2012). Além disso, TITTON (2006) já havia observado este baixo desempenho do Windows, em comparação ao Linux, quando utilizada a plataforma de grid Condor.

6 CONSIDERAÇÕES FINAIS

O poder de processamento dos computadores pessoais segue aumentando. Assim como os computadores pessoais, os dispositivos móveis apresentam um maior poder de processamento, tornando-se cada vez mais atrativos para uso em processamentos de alto desempenho. Como resposta a este crescimento, houve o desenvolvimento de tecnologias que permitiram a utilização do poder de processamento destes dispositivos em conjunto, de forma acelerar a execução de aplicações. A plataforma BOINC, que já fornecia uma infraestrutura para desenvolvimento de aplicações de computação pública, passou a suportar oficialmente o sistema operacional Android, de modo a possibilitar que um mesmo projeto possa utilizar computadores voluntários que possuam quatro sistemas operacionais bastante utilizados atualmente: Windows, Linux, MacOS e Android (BOINC Documentation, 2015).

A aplicação escolhida para a validação de uma implementação nas plataformas Windows, Linux e Android foi o cálculo dos dígitos hexadecimais do Pi. A fim de garantir que as tarefas pudessem ser executadas de modo independente, um algoritmo de extração de dígitos foi escolhido para ser implementado. Durante a pesquisa, duas opções de algoritmos de extração de dígitos foram verificadas: a fórmula BBP e a fórmula de Bellard. A tentativa de implementação da fórmula de Bellard falhou em função da pouca documentação encontrada a respeito desta fórmula. Por este motivo, a fórmula BBP foi implementada neste trabalho.

Em função deste tipo de aplicação exigir um alto grau de precisão nos valores decimais, que estão sujeitos à falhas nas representações computacionais de ponto-flutuante, foi necessário o uso de uma biblioteca para cálculos de precisão arbitrária. A biblioteca utilizada para esta finalidade foi a GNU MPFR, por permitir seleção de modo de arredondamento e quantidade arbitrária de bits de precisão.

Através do desenvolvimento do presente projeto, foi possível confirmar a possibilidade de desenvolvimento de uma aplicação de computação pública para cálculo dos dígitos do Pi com suporte a dispositivos móveis. Esta aplicação contou com um algoritmo simples e eficiente computacionalmente, que calculou 10000 dígitos do Pi utilizando a plataforma BOINC. Os valores calculados foram validados a partir de

uma comparação com os resultados fornecidos pelo projeto HexPi (HexPi, 2015).

Para a avaliação de desempenho, foi utilizado o cálculo do dígito 100.000 do Pi. A partir da análise dos resultados, pôde-se perceber que o melhor desempenho foi obtido quando a execução ocorreu no sistema operacional Linux. Já a plataforma Windows apresentou desempenho inferior aos demais dispositivos. A diferença de desempenho obtida nas execuções em sistemas operacionais Windows e Linux corrobora aquela que já havia sido observada por SMIDERLE (2012) e TITTON (2006).

No que diz respeito aos dispositivos com sistema operacional Android, apesar de sua reduzida capacidade de processamento em relação aos demais computadores utilizados, seu desempenho apresentou-se mais satisfatório que o desempenho do sistema operacional Windows.

6.1 Trabalhos Futuros

Como sugestões de trabalhos futuros, destacam-se: a implementação do suporte ao sistema operacional MacOS; o suporte às demais arquiteturas do sistema operacional Android (MIPS e x86); a compilação de uma versão que suporte o sistema operacional Linux 32-bit. Podemos citar também um estudo aprofundado da aplicação para identificação das peculiaridades que causaram a diferença de desempenho observada no sistema operacional Windows em relação ao Linux.

Além disso, podemos citar a modificação do programa para utilizar o algoritmo da fórmula de Bellard, ao invés da fórmula BBP que foi implementada neste projeto. Pode-se ainda sugerir a mudança da aplicação para receber o dígito a ser calculado pela linha de comando ao invés de receber através de um arquivo, como ocorre atualmente. Por fim, um trabalho futuro poderia implementar a utilização do mecanismo de *checkpoints* do BOINC para garantir que, em caso de interrupção, a WU seja inicializada a partir do ponto de interrupção.

REFERÊNCIAS

ABC@Home. <Disponível em: http://abcathome.com/>. Acesso em 12 de março de 2015.

ANDERSON, D.; FEDAK, G. The Computational and Storage Potential of Volunteer Computing. In: CLUSTER COMPUTING AND THE GRID, 2006. CCGRID 06. SIXTH IEEE INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2006. v.1, p.73–80.

ANDERSON, D. P. BOINC: a system for public-resource computing and storage. In: IEEE/ACM INTERNATIONAL WORKSHOP ON GRID COMPUTING, 5., Washington, DC, USA. **Proceedings...** IEEE Computer Society, 2004. p.4–10. (GRID '04).

Background Pi. <Disponível em: https://www.codingwell.net/projects/>. Acesso em 12 de março de 2015.

BAILEY, D. H. **The BBP Algorithm for Pi**. <Disponível em: http://www.davidhbailey.com/dhbpapers/bbp-alg.pdf>. Acesso em 07 de abril de 2015.

BELLARD, F. **A new formula to compute the nth digit of Pi**. <Disponível em: http://bellard.org/pi/pi_bin/pi_bin.html>. Acesso em 25 de abril de 2015.

BOINC Documentation. <Disponível em: http://boinc.berkeley.edu/trac/wiki/ProjectMain>. Acesso em 19 de março de 2015.

Folding@Home. <Disponível em: http://folding.stanford.edu/>. Acesso em 12 de março de 2015.

GRANLUND, T. **GMP**: the gnu multiple precision arithmetic library. 4.1.2.ed. [S.l.]: The Free Software Foundation, 2002.

HexPi. <Disponível em: http://hexpi.sourceforge.net/>. Acesso em 20 de junho de 2015.

LHC@Home. <Disponível em: http://lhcathome.web.cern.ch/>. Acesso em 12 de março de 2015.

LIMA, R. L. Q. de. **Capítulo II**: funções lineares. função modular. função piso. <Disponível em: http://www.uff.br/webmat/Calc1_LivroOnLine/Cap02_Calc1.html>. Acesso em 29 de junho de 2015.

MCKEEMAN, B. **The Computation of Pi by Archimedes**. <Disponível em: http://www.mathworks.com/matlabcentral/fileexchange/29504-the-computation-of-pi-by-archimedes/content/html/ComputationOfPiByArchimedes.html#28>. Acesso em 09 de maio de 2015.

MIRANDA, D. de. **Cálculo do valor Pi**. <Disponível em: http://educador.brasilescola.com/estrategias-ensino/calculo-valor-pi.htm>. Acesso em 17 de abril de 2015.

Monte Carlo Pi. <Disponível em: http://mathfaculty.fullerton.edu/mathews/n2003/montecarlopimod.html>. Acesso em 06 de abril de 2015.

MYSQL. **MySQL Community Server**. <Disponível em: https://dev.mysql.com/downloads/mysql/>. Acesso em 01 de julho de 2015.

PARSONS, J. J.; OJA, D. **New Perspectives on Computer Concepts 2014**: comprehensive. 16th.ed. [S.l.]: Cengage Learning, 2014.

PiHex Project. <Disponível em http://oldweb.cecm.sfu.ca/projects/pihex/index.html>. Acesso em 02 de março de 2015.

PrimeGrid. <Disponível em: http://www.primegrid.com/>. Acesso em 12 de março de 2015.

SANTOS, A. A. da Luz dos. **Reconhecimento de Padrões em DNA Utilizando Computação Pública**. Caxias do Sul, Brasil: Universidade de Caxias do Sul, 2005.

SETI@Home. <Disponível em: http://setiathome.berkeley.edu/>. Acesso em 12 de março de 2015.

SHENGDA, Z. et al. The Calculation and Analysis of PI in Computer Numerical Simulation. In: DIGITAL MANUFACTURING AND AUTOMATION (ICDMA), 2012 THIRD INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2012. p.887–890.

SILVEIRA, J. F. P. da. **Cálculo das Constantes Elementares Clássicas**: o caso do pi. <Disponível em: http://www.mat.ufrgs.br/~portosil/aplcom1a.html>. Acesso em 02 de março de 2015.

SMIDERLE, R. **Computação Pública para Dispositivos Móveis Baseados em Android**. Caxias do Sul, Brasil: Universidade de Caxias do Sul, 2012.

SMITH, J. O. **Physical Audio Signal Processing**. <Disponível em: https://ccrma.stanford.edu/~jos/pasp/Arctangent_Series_Expansion.html>. Acesso em 10 de maio de 2015.

The Apache Foundation. **Apache HTTP Server Project**. <Disponível em: http://httpd.apache.org/>. Acesso em 01 de julho de 2015.

TITTON, J. **Processamento de Eletroencefalograma utilizando o método Matching Pursuit em uma plataforma Grid**. Caxias do Sul, Brasil: Universidade de Caxias do Sul, 2006.

Twin Prime Search. <Disponível em: http://primes.utm.edu/bios/page.php?id=949>. Acesso em 12 de março de 2015.

WANG, D. et al. CampusWare: an easy-to-use, efficient and portable grid middleware for compute-intensive applications. In: CHINAGRID ANNUAL CONFERENCE, 2009. CHINAGRID '09. FOURTH. **Anais...** [S.l.: s.n.], 2009. p.36–43.

WEISSTEIN, E. W. **CRC Concise Encyclopedia of Mathematics**. 2nd.ed. [S.l.]: Chapman & Hall/CRC Press, 2002.

WEISSTEIN, E. W. **Pi Formulas**. <Disponível em: http://mathworld.wolfram.com/PiFormulas.html>. Acesso em 29 de junho de 2015.

ANEXO A - CÓDIGO-FONTE DO PROJETO

7.1 Aplicação principal

Quadro 7.1: bbp_pi.h - definições das funções da aplicação.

```
1  #ifndef BBP_PI_H
2  #define BBP_PI_H
3
4  #include <mpfr.h>
5  #include <limits.h>
6  #include <math.h>
7
8  #include "boinc_api.h"
9  #include "filesys.h"
10 #include "util.h"
11
12 #define PRECISION 2048
13 #define BASE 16UL
14
15 /*
16  * Binary exponentiation modulo k algorithm, as written by David H. Bailey.
17  */
18 void expm(mpfr_t, double, double);
19
20 /*
21  * Partial series for calculating the BBP formula.
22  */
23 void partial_bbp(mpfr_t, int, int);
24
25 /*
26  * Hexadecimal translation function.
27  */
28 void hexstr(mpfr_t value, int digits, char str[]);
29
30 /*
31  * Main Pi calculation and hex translation.
32  */
33 void pi_value(char str[], const int dignum, long position);
34
35 #endif //BBP_PI_H
```

Quadro 7.2: bbp_pi.cpp - implementação das funções da aplicação.

```
1  #include "bbp_pi.h"
2
3  /*
4   * Binary exponentiation modulo k algorithm, as written by David H. Bailey.
5   */
6  void expm(mpfr_t rop, double dn, double dk) {
7          mpfr_t t;
8          mpfr_t r;
9          mpfr_t n;
10         mpfr_t k;
11         mpfr_t mid_calc;
12
13         mpfr_init(t);
14         mpfr_init(r);
15         mpfr_init(mid_calc);
16         mpfr_init_set_d(n, dn, MPFR_RNDN);
17         mpfr_init_set_d(k, dk, MPFR_RNDN);
18
19         mpfr_log2(t, n, MPFR_RNDN);
20         mpfr_trunc(t, t);
21         mpfr_exp2(t, t, MPFR_RNDN);
22
23         mpfr_set_d(r, 1.f, MPFR_RNDN);
24
25         if(mpfr_cmp_d(k, 1.f) == 0)
26                 mpfr_set_d(r, 0.f, MPFR_RNDN);
27     else
28                 while(42) {
29                         if(mpfr_cmp(n, t) >= 0) {
30                                 mpfr_mul_d(r, r, BASE, MPFR_RNDN);
31                                 mpfr_div(mid_calc, r, k, MPFR_RNDN);
32                                 mpfr_trunc(mid_calc, mid_calc);
33                                 mpfr_mul(mid_calc, mid_calc, k, MPFR_RNDN);
34                                 mpfr_sub(r, r, mid_calc, MPFR_RNDN);
35                                 mpfr_sub(n, n, t, MPFR_RNDN);
36                         }
37
38                         mpfr_mul_d(t, t, 0.5f, MPFR_RNDN);
39                         mpfr_set_d(mid_calc, .0f, MPFR_RNDN);
40
41                         if(mpfr_cmp_d(t, 1.f) >= 0) {
42                                 mpfr_mul(r, r, r, MPFR_RNDN);
43                                 mpfr_div(mid_calc, r, k, MPFR_RNDN);
44                                 mpfr_trunc(mid_calc, mid_calc);
45                                 mpfr_mul(mid_calc, mid_calc, k, MPFR_RNDN);
46                                 mpfr_sub(r, r, mid_calc, MPFR_RNDN);
47                         } else
48                                 break;
49                 }
50
51         mpfr_set(rop, r, MPFR_RNDN);
52         mpfr_clears(t, r, mid_calc, n, k, (mpfr_ptr) NULL);
53 }
54
55 /*
56  * Partial series for calculating the BBP formula.
57  */
58 void partial_bbp(mpfr_t rop, int d, int j) {
```

```
59          mpfr_t value;
60          mpfr_t mid_calc;
61          int div;
62          int k;
63
64          mpfr_init_set_d(value, 0.0f, MPFR_RNDN);
65          mpfr_init(mid_calc);
66
67          for(k = 0;k < d;k++) {
68                  div = 8 * k + j;
69                  expm(mid_calc, d - k, div);
70                  mpfr_div_si(mid_calc, mid_calc, div, MPFR_RNDN);
71                  mpfr_add(value, value, mid_calc, MPFR_RNDN);
72                  mpfr_trunc(mid_calc, value);
73                  mpfr_sub(value, value, mid_calc, MPFR_RNDN);
74          }
75
76          for(k = d;k <= d + 100;k++) {
77                  div = 8 * k + j;
78                  mpfr_add_d(value, value, pow(BASE, d - k)/div, MPFR_RNDN);
79          }
80
81          mpfr_trunc(mid_calc, value);
82          mpfr_add_d(mid_calc, mid_calc, 1.f, MPFR_RNDN);
83          mpfr_sub(value, value, mid_calc, MPFR_RNDN);
84
85          mpfr_init_set(rop, value, MPFR_RNDN);
86          mpfr_clears(value, mid_calc, (mpfr_ptr) NULL);
87  }
88
89  /*
90   * Hexadecimal translation function.
91   */
92  void hexstr(mpfr_t value, int digits, char str[]) {
93          const char hexdigits[] = "0123456789ABCDEF";
94          int idx;
95          mpfr_t hexval;
96          mpfr_t mid_calc;
97
98          mpfr_init(hexval);
99          mpfr_init(mid_calc);
100         mpfr_abs(hexval, value, MPFR_RNDN);
101
102         for(idx = 0;idx < digits;idx++) {
103                 mpfr_mul_d(hexval, hexval, BASE, MPFR_RNDN);
104                 mpfr_trunc(mid_calc, hexval);
105                 str[idx] = hexdigits[mpfr_get_ui(mid_calc, MPFR_RNDN)];
106                 mpfr_sub(hexval, hexval, mid_calc, MPFR_RNDN);
107         }
108
109         str[idx] = '\0';
110         mpfr_clears(hexval, mid_calc, (mpfr_ptr) NULL);
111 }
112
113 /*
114  * Main Pi calculation and hex translation.
115  */
116 void pi_value(char str[], const int dignum, long position) {
117         mpfr_t value;
```

```
118         mpfr_t part1;
119         mpfr_t part2;
120         mpfr_t part3;
121         mpfr_t part4;
122         mpfr_t mid_calc;
123         double d = (double) position;
124 
125         mpfr_set_default_prec(PRECISION);
126 
127         mpfr_init(value);
128 
129         partial_bbp(part1, d, 1);
130         partial_bbp(part2, d, 4);
131         partial_bbp(part3, d, 5);
132         partial_bbp(part4, d, 6);
133 
134         mpfr_mul_d(part1, part1, 4.f, MPFR_RNDN);
135         mpfr_mul_d(part2, part2, 2.f, MPFR_RNDN);
136 
137         mpfr_set(value, part1, MPFR_RNDN);
138         mpfr_sub(value, value, part2, MPFR_RNDN);
139         mpfr_sub(value, value, part3, MPFR_RNDN);
140         mpfr_sub(value, value, part4, MPFR_RNDN);
141 
142         mpfr_init_set(mid_calc, value, MPFR_RNDN);
143         mpfr_trunc(mid_calc, mid_calc);
144         mpfr_sub(value, value, mid_calc, MPFR_RNDN);
145         mpfr_add_d(value, value, 1.f, MPFR_RNDN);
146         mpfr_floor(mid_calc, value);
147         mpfr_sub(value, value, mid_calc, MPFR_RNDN);
148 
149         hexstr(value, dignum, str);
150 
151         mpfr_clear(value);
152         mpfr_clear(part1);
153         mpfr_clear(part2);
154         mpfr_clear(part3);
155         mpfr_clear(part4);
156         mpfr_clear(mid_calc);
157 }
```

Quadro 7.3: prc_pi.cpp - ponto de entrada da aplicação.

```
1  #include <stdio.h>
2 
3  #include "bbp_pi.h"
4 
5  #define DIGITS_NUMBER 10
6 
7  const int filename_length = 8192;
8 
9  /*
10  * Reads the file containing the digit position to compute.
11  */
12 static long read_digit_position() {
13         char filename[filename_length];
14         int retval;
15         long position;
16         FILE *input;
```

```
17
18          retval = boinc_resolve_filename("infile",
19                          filename,
20                          filename_length);
21
22          if(retval) {
23                  perror("couldn't open file");
24                  exit(-1);
25          }
26
27          input = boinc_fopen(filename, "r");
28          fscanf(input, "%ld", &position);
29          fclose(input);
30
31          return position;
32  }
33
34  /*
35   * Writes the resulting value in the output file.
36   */
37  static void write_output_file(char hexstr[]) {
38          char filename[filename_length];
39          int retval;
40          FILE *output;
41
42          retval = boinc_resolve_filename("outfile",
43                          filename,
44                          filename_length);
45
46          if(retval) {
47                  perror("couldn't open file");
48                  exit(-1);
49          }
50
51          output = boinc_fopen(filename, "w");
52          fputs(hexstr, output);
53          fclose(output);
54  }
55
56  /*
57   * Calls functions to do effective Pi calculation.
58   */
59  static void calculate_pi() {
60          char hexdigits[DIGITS_NUMBER + 1];
61          long position;
62
63          position = read_digit_position();
64
65          printf("digit position: %d\n", position);
66
67          pi_value(hexdigits, DIGITS_NUMBER, position);
68
69          write_output_file(hexdigits);
70  }
71
72  int main(int argc, char **argv) {
73      char buf[256];
74      int retval = boinc_init();
75      if (retval) {
```

```
76          fprintf(stderr, "%s boinc_init returned %d\n",
77              boinc_msg_prefix(buf, sizeof(buf)), retval
78          );
79          exit(retval);
80      }
81
82          calculate_pi();
83
84          return boinc_finish(0);
85  }
86
87  #ifdef _WIN32
88  int WINAPI WinMain(
89      HINSTANCE hInst, HINSTANCE hPrevInst, LPSTR Args, int WinMode
90  ) {
91      LPSTR command_line;
92      char* argv[100];
93      int argc;
94
95      command_line = GetCommandLine();
96      argc = parse_command_line(command_line, argv);
97      return main(argc, argv);
98  }
99  #endif
```

7.1.1 Arquivo de compilação para Linux 64-bit

Quadro 7.4: Makefile.app

```
1   CXX = g++
2   compiler_flags = -Wall -pedantic
3   math_flags = -lgmp -lmpfr -lm -lstdc++
4   boinc_flags = -lboinc_api -lboinc -lboinc_crypt -lssl -lcrypto
5   linker_flags = -L/usr/local/lib
6   include_flags = -I/usr/local/include/boinc
7
8   build:
9           $(CXX) \
10                  $(compiler_flags) -O2 -O3 prc_pi.cpp bbp_pi.cpp\
11                  $(include_flags) -o prc_pi\
12                  $(linker_flags) \
13                  $(math_flags) \
14                  $(boinc_flags)
15
16  debug:
17          $(CXX) \
18                  $(compiler_flags) -g prc_pi.cpp bbp_pi.cpp\
19                  $(include_flags) -o prc_pi\
20                  $(linker_flags) \
21                  $(math_flags) \
22                  $(boinc_flags)
```

7.1.2 Arquivo de compilação para Android ARM

Quadro 7.5: Makefile.droid

```
MPFRDIR = $(HOME)/Documents/mpfr/mpfr-lib-android15
GMPDIR =  $(HOME)/Documents/mpfr/gmp-lib
BOINCDIR = $(HOME)/androidarm-tc/arm-linux-androideabi
NDKCDIR = /opt/android-ndk-r10e/platforms/android-15/arch-arm
LGCCDIR = $(HOME)/androidarm-tc/lib/gcc/arm-linux-androideabi/4.8
GCCFLAGS = -Wl,-rpath -Wl,$(LIBDIR) \
                    -I$(MPFRDIR)/include \
                    -I$(GMPDIR)/include \
                    -I$(BOINCDIR)/include \
                    -I$(BOINCDIR)/include/boinc \
                    --sysroot=$(NDKCDIR)
LDFLAGS = -L$(MPFRDIR)/lib \
                   -L$(GMPDIR)/lib \
                   -L$(BOINCDIR)/lib \
                   -L$(LGCCDIR) \
                   -Wl,-Bstatic -lmpfr -lgmp -Wl,-Bdynamic -lboinc_api -lboinc -lgcc -
                       Wl,--as-needed

build:
        $(CXX) bbp_pi.cpp prc_pi.cpp -o prc_pi_androidarm \
                $(GCCFLAGS) \
                $(LDFLAGS)

clean:
        rm prc_pi
```

7.2 Assimilador de resultados

Quadro 7.6: Implementação do assimilador.

```
#ifndef ASSIMILATE_HANDLER_H
#define ASSIMILATE_HANDLER_H

#include <assimilate_handler.h>
#include <boinc_db.h>
#include <error_numbers.h>
#include <filesys.h>
#include <sched_msgs.h>
#include <validate_util.h>
#include <sched_config.h>
#include <sched_util.h>
#include <mpfr.h>
#include <errno.h>

using std::vector;

#define CHARNUMBER 11
#define BASE 16UL
#define DIGIT_POS 9

int write_error(char* p) {
    static FILE* f = 0;
    if (!f) {
```

```
24          f = fopen(config.project_path("sample_results/errors"), "a");
25          if (!f) return ERR_FOPEN;
26      }
27      fprintf(f, "%s", p);
28      fflush(f);
29      return 0;
30  }
31
32  int assimilate_handler(WORKUNIT &wu,
33                  vector<RESULT> &results,
34                  RESULT &canonical_result) {
35          std::string filepath;
36          char wunumber[20];
37          char buf[1024];
38          FILE *result_file;
39          FILE *output;
40          char hexstr[CHARNUMBER];
41          mpfr_t value;
42          mpfr_t term;
43          int i;
44          int index = -1;
45          long digit;
46          int retval;
47
48          if(wu.canonical_resultid) {
49                  retval = boinc_mkdir(config.project_path("wu_results"));
50                  if(retval) {
51                          return retval;
52                  }
53
54                  strcpy(wunumber,
55                                  &wu.name[DIGIT_POS]);
56
57                  get_output_file_path(canonical_result,
58                                  filepath);
59                  result_file = fopen(filepath.c_str(),
60                                  "r");
61
62                  if(!result_file) {
63                          perror("error opening result file");
64                          return errno;
65                  }
66
67                  fgets(hexstr,
68                                  CHARNUMBER,
69                                  result_file);
70
71                  mpfr_init_set_si(value,
72                                  0L,
73                                  MPFR_RNDN);
74
75                  for(i = 0;i < CHARNUMBER - 1;i++) {
76                          if(hexstr[i] >= 'A' && hexstr[i] <= 'F') {
77                                  digit = hexstr[i] - 'A' + 10;
78                          } else {
79                                  digit = hexstr[i] - '0';
80                          }
81
82                          mpfr_init_set_ui(term,
```

```
                                        BASE,
                                        MPFR_RNDN);
                        mpfr_pow_si(term,
                                        term,
                                        index,
                                        MPFR_RNDN);
                        mpfr_mul_si(term,
                                        term,
                                        digit,
                                        MPFR_RNDN);

                        index--;

                        mpfr_add(value,
                                        value,
                                        term,
                                        MPFR_RNDN);

                        mpfr_clear(term);
                }

                output = fopen(config.project_path("wu_results/results"),
                                "a");
                fprintf(output,
                                "%10d,\"%s\",%.2000f\n",
                                atoi(wunumber),
                                hexstr,
                                mpfr_get_d(value,
                                        MPFR_RNDN));
                fflush(output);
                fclose(output);

                mpfr_clear(value);
                fclose(result_file);
        } else {
        sprintf(buf, "%s: 0x%x\n", wu.name, wu.error_mask);
        return write_error(buf);
        }

        return 0;
}

#endif // ASSIMILATE_HANDLER_H
```

7.2.1 Arquivo de compilação do assimilador para Linux 64-bit

Quadro 7.7: Makefile.assimilator

```
GCCFLAGS = -I$(HOME)/Documents/boinc-pi/boinc-v2 \
                    -I$(HOME)/Documents/boinc-pi/boinc-v2/lib \
                    -I$(HOME)/Documents/boinc-pi/boinc-v2/db \
                    -I$(HOME)/Documents/boinc-pi/boinc-v2/sched \
                    -I/usr/include/mysql
LDFLAGS = -L/usr/lib/mysql -lgmp -lmpfr

build:
        g++ -Wall -g \
                assimilate_handler.cpp \
```

```
11                -o assimilator \
12                $(GCCFLAGS) \
13                $(LDFLAGS)
```

7.3 Gerador de *workunits*

Quadro 7.8: Implementação do gerador de *workunits*.

```
1  #include <boinc_db.h>
2  #include <backend_lib.h>
3  #include <util.h>
4
5  /**
6   * Writes input file to download directory.
7   */
8  void write_input_file(const char *path,
9                  const void *contents,
10                 const size_t size,
11                 const size_t count) {
12         FILE *f = fopen(path, "w");
13
14         if(f == NULL) {
15                 perror("Error opening input file");
16         } else {
17                 if(fwrite(contents,
18                                 size,
19                                 count,
20                                 f) < count) {
21                         fprintf(stderr,
22                                         "Error writing input file %s.\n",
23                                         path);
24                 }
25
26                 fclose(f);
27         }
28 }
29
30 /**
31  * Generates workunits.
32  */
33 void generate_workunits(long first_digit, long last_digit) {
34         DB_APP app;
35         DB_WORKUNIT wu;
36         char *wu_template;
37         const char *infiles[1];
38         char path[1024];
39         SCHED_CONFIG config;
40         long digit = 0;
41         char data[200];
42         char wuname[400];
43         char filename[256];
44
45         infiles[0] = filename;
46
47         config.parse_file();
48
49         boinc_db.open(config.db_name,
```

```
                        config.db_host,
                        config.db_user,
                        config.db_passwd);
        app.lookup("where name='prc_pi'");

        read_file_malloc("templates/input",
                        wu_template);

        for(digit = first_digit;digit <= last_digit;digit++) {
                sprintf(data,
                                "%ld",
                                digit);
                sprintf(wuname,
                                "prcpi_wu_%ld",
                                digit);
                sprintf(filename,
                                "wu_input%ld",
                                digit);

                config.download_path(filename,
                                path);

                write_input_file(path,
                        data,
                        sizeof(char),
                        strlen(data));

                wu.clear();
                strcpy(wu.name,
                                wuname);
                wu.appid = app.id;

                create_work(wu,
                                wu_template,
                                "templates/output",
                                "templates/output",
                                infiles,
                                1,
                                config);
        }
}

int main(int argc, char **argv) {
        long first_digit;
        long last_digit;

        if(argc < 3) {
                printf("Usage: workunit_maker first_digit last_digit\n");
                return 0;
        }

        first_digit = atol(argv[1]);
        last_digit = atol(argv[2]);

        if(first_digit > last_digit) {
                printf("last_digit must be bigger than or equal to first_digit\n");
                return 0;
        }

```

```
109         generate_workunits(first_digit,
110                            last_digit);
111         return 0;
112 }
```

7.3.1 Arquivo de compilação do gerador de *workunits* para Linux 64-bit

Quadro 7.9: Makefile.wugen

```
1  CXX=g++
2  CXXFLAGS=-pthread -Wall -g
3  MYSQL_FLAGS=-lmysqlclient -lpthread -lz -lm -lssl -lcrypto -ldl
4  BOINC_FLAGS=-lboinc_api -lsched -lboinc -lboinc_crypt
5  INCLUDE_FLAGS=-I/usr/local/include/boinc -I/usr/include/mysql
6  LINKER_FLAGS=-L/usr/local/lib -L/usr/lib64/mysql
7
8  build:
9          $(CXX) workunit_maker.cpp\
10                 $(CXXFLAGS) \
11                 $(INCLUDE_FLAGS) \
12                 $(LINKER_FLAGS) \
13                 $(BOINC_FLAGS) \
14                 $(MYSQL_FLAGS) \
15                 -o workunit_maker
16
17 copy:
18         cp -v workunit_maker /boinc/prc-pi/bin
19
20 all: build copy
```