PMC-581 Projeto Mecânico II

JPM - Java Parallel Machine

Um Sistema de Computação Distribuída para Reconhecimento de Impressões Digitais

10,0 (dez)

Orientador: Prof. Dr. Newton Maruyama

Alvaro Guimarães Mucida

Reynaldo Penharrubia Fagundes

Dezembro 1998

Agradecimentos

Ao Professor Newton Maruyama, pela dedicação espartana e indispensável contribuição para a realização deste trabalho.

Aos Professores desta Escola, que proporcionaram anos de aprendizado e amadurecimento pessoal, com conhecimentos que não nos faltarão nas etapas ainda por vir de nossas vidas.

Aos nossos colegas, alunos e alunas desta Escola, por sua amizade e fraternidade que tornaram suave a jornada que agora concluímos.

À todos aqueles que, algum dia, ofereceram seu tempo e dedicação e de alguma forma contribuiram para a realização deste que foi nosso maior objetivo, em cinco dos nossos melhores anos de vida.

Sumário

Resumo

A necessidade de identificação tem crescido nos últimos tempos, em face à cada vez maior automação de tarefas que dela dependem, notadamente no setor bancário, policial e de segurança genérica de sistemas.

Além dessa, nos últimos 3 anos, observa-se uma convergência de tecnologias paralelas, tais como o desenvolvimento de linguagens de programação "universais" e multi-plataforma, popularização das redes de computadores em ambientes corporativos, crescimento das mega-redes heterogêneas (como a Internet) e aumento da capacidade computacional dos chamados "computadores de mesa" (Desktop's), permitindo o surgimento de sistemas de computação distribuída visando processamentos computacionalmente onerosos.

Este projeto visa o desenvolvimento de um sistema de identificação pessoal, baseado em reconhecimento de impressões digitais, usando computação paralela em plataformas de baixo custo para levar à cabo os processamentos excessivamente custosos associados ao reconhecimento de padrões.

A "Java Parallel Machine" (ou JPM) é uma máquina de computação paralela virtual, baseada em linguagem Java, multi-plataforma, que se propõe a realizar o reconhecimento de impressões digitais, utilizando uma ténica que utliza lógica Fuzzy. As etapas computacionalmente custosas desta tarefa são implementadas usando uma biblioteca que permite lançar mão de várias máquinas de uma rede para construir um "computador paralelo virtual" e assim, viabilizar os tempos de processamento impraticáveis sem o uso de computação paralela.

A implementação visa demonstrar a viabilidade do uso, hoje, de técnicas de computação distribuída em redes corporativas e heterogêneas, subsidiando argumentos otimistas com relação à viabilidade de implantação de tais sistemas em redes "reais", e oferecendo contrapontos àqueles

excessivamente otimistas, como os que acreditam na viabilidade imediata de processamento distribuído em redes heterogêneas e de baixa confiabilidade, como a Internet.

O sistema gerado apresentou um comportamento excelente com relação às questões relativas à computação paralela (curva de speed-up bastante próxima da teoricamente ideal e com boa porção linear, baixo overhead de comunicação para aplicações em Intranet's, principalmente) e surpreendentemente mostrou-se também muito confiável quanto ao reconhecimento, propriamente dito, de digitais (probabilidade de falsa aceitação menor do que 0,1%).

1. Introdução

A habilidade que certos sistemas computacionais têm de realizar suas tarefas (genericamente, a solução de um problema qualquer) subdividindo a tarefa inicial em subtarefas a serem realizadas em diferentes unidades de processamento ao mesmo tempo é usualmente chamada de computação distribuída. Lançando mão desta capacidade, é possível aproveitar os recursos do que podemos chamar de "computadores paralelos" de forma eficiente, já que temos as várias unidades de processamento realizando suas tarefas de maneira simultânea e parcialmente independente (já que, tipicamente, as subtarefas não são totalmente independentes umas das outras, mas sim ligadas por uma hierarquia de execução).

Certamente, o uso da computação paralela é desejável quando se percebe que a arquitetura dos modernos computadores esbarra em barreiras tecnológicas que tornam mais viável (normalmente por motivos econômicos) criar computadores com duas ou mais unidades de processamento do que computadores com recursos de processamento maiores ou melhores.

Como quase sempre acontece com os avanços tecnológicos que se tornam lugar comum nos dias modernos, o surgimento de máquinas com processamento paralelo se deu no meio acadêmico, e por muito tempo a computação distribuída se resumia ao conjunto de ferramentas e técnicas utilizadas para operar os então "computadores paralelos", máquinas dotadas de mais de um processador. Com o tempo, entretanto, novos paradigmas surgiram e aproximaram cada vez mais este tipo de sistema do uso em ambientes corporativos.

Nos dias atuais, entretanto, percebe-se um notável avanço do tema. O antigo modelo de "computadores paralelos" vem sendo gradativamente repensado para sistemas de computação distribuída, onde se tem vários computadores, eventualmente heterogêneos, constituindo uma "força computacional" para a resolução do problema, e não um único computador

paralelo. Esse tipo de ambiente mostra-se muito robusto já que se pode, a baixo custo, constituir uma grande "máquina paralela" formada por centenas ou mesmo milhares de máquinas "simples", computadores já existentes, por exemplo, em redes corporativas de grandes corporações. Essa nova realidade vem estimular o uso de soluções de computação que se utilizem de um grande número de computadores, heterogêneos, que se comunicam por algum tipo de rede e eventualmente individualmente falíveis (o que torna necessários modelos regenerativos tolerantes a falhas) para a realização de tarefas complexas.

Desta forma, a computação distribuída como uma alternativa de computação de alto desempenho barata e flexível, em ambientes (cada vez mais comuns) de redes com imenso número de máquinas hoje em dia dotadas de razoável grau de processamento (computadores, por exemplo, com arquitetura Intel, abundantes e normalmente subutilizados em Intranet's de grandes corporações), conectadas por redes cada vez mais velozes e, ainda, com o surgimento de novos modelos de programação universais (dentre os quais destaca-se a linguagem Java) nos encorajam à direções ainda pouco exploradas e incrivelmente promissoras.

Naturalmente, para que todas estas características da computação distribuída pudessem ser implementadas, testadas e apresentadas, uma aplicação exemplo teve que ser eleita. Esta aplicação teria que apresentar características adaptáveis à sua implementação em computação distribuída (como executar tarefas computacionalmente intensivas e que de certa forma pudessem ser distribuídas de forma razoavelmente independente entre diferentes unidades de processamento) e também fazer parte do rol de tecnologias em acentuado desenvolvimento e incontestável importância, de forma que se pudesse unir em um só trabalho dois recentes e modernos campos de pesquisa científica.

As Tecnologias Biométricas, que para o presente trabalho, aparecem como instrumento para sistemas de identificação pessoal, poderiam ser definidas como métodos automatizados para verificação e/ou reconhecimento

da identidade de pessoas baseados em características fisiológicas e/ou comportamentais. Na grande maioria dos casos, sistemas biométricos de reconhecimento e identificação pessoais envolvem: (i) mecanismos para aquisição e captura de uma imagem analógica ou digital de características pessoais de uma pessoa; (ii) compressão, processamento e comparação da imagem; (iii) interface com as aplicações dos sistemas. Vale fazer uma distinção clara entre as características fisiológicas e comportamentais: as primeiras são características relativamente estáveis, como uma impressão digital, a forma e o contorno das mãos, os padrões da íris, ou os padrões dos vasos sangüíneos na pupila; as últimas são mais um reflexo do perfil psicológico da pessoa, ainda que fatores como sexo ou tamanho tenham influência considerável. A assinatura é um exemplo clássico de característica comportamental utilizada para reconhecimento e identificação, assim como a maneira como uma pessoa digita ou a forma de falar.

São características utilizadas em sistemas biométricos automatizados de identificação: impressões digitais, padrões nos olhos, contorno e forma das mãos, a assinatura, a voz, a dinâmica da digitação e características faciais, entre outros. A característica eleita como objeto de estudo do presente trabalho é a impressão digital, cuja estabilidade e unicidade estão estabelecidas e comprovadas no meio científico. Sob cuidadosa investigação, estima-se que a chance de duas pessoas terem impressões digitais consideradas idênticas seja de uma em um bilhão, incluindo gêmeos, o que prova a eficácia do método para os fins a que se propõe.

O uso de impressões digitais para fins de identificação pessoal tem se revelado um método promissor para no futuro ampliar razoavelmente sua base de aplicação e quiçá substituir muitas formas já consagradas de identificação pessoal como senhas ou cartões. Por outro lado, além do controle de acesso (seja a locais, ou sistemas ou qualquer outra entidade, física ou lógica), as impressões digitais também tem grande importância para as polícias e/ou quaisquer instituições que por algum motivo podem desejar identificar uma pessoa com alto grau de confiabilidade.

Assim propõe-se aqui o desenvolvimento de um sistema que consiga unir da melhor forma possível a tecnologia da computação distribuída com o uso de impressões digitais para o reconhecimento de identidade, aproveitando todas as particularidades e sinergias entre estes dois recentes ramos do conhecimento humano.

2. Objetivos

Os principais objetivos deste projeto são o estudo, a concepção e a implementação de um Sistema de Computação Distribuída para o Reconhecimento de Impressões Digitais. Deve-se enfatizar que este Sistema de Computação Distribuída deve ser flexível e potencialmente adaptável para outras aplicações que não apenas a extração de características representativas de impressões digitais e para tanto deve ser desenvolvida uma biblioteca para o gerenciamento de processamento distribuído tão genérica quanto possível.

Além disso, o presente projeto também compreende a fase de testes do sistema proposto sob diversas condições e rodando em ambientes heterogêneos, observando-se a sensibilidade do sistema ao número de processadores, às condições de rede, simulações de falha, observando-se também os problemas de "*overhead*" de comunicação e distribuição de carga.

O algoritmo para a extração de características representativas de impressões digitais deve ainda ser implementado de forma a obter ganho de performance em ambientes mais potentes, isto é, deve possuir tarefas computacionalmente custosas que possam ser distribuídas de maneira relativamente independente.

3. Especificação do Problema

Para que os objetivos do presente projeto fossem atingidos, separou-se as diferentes características que deveriam ser especificadas para que cada uma fosse tratada individualmente e baseado em critérios diferentes para cada uma delas se chegasse a uma decisão quanto a melhor maneira de implementar cada uma das características do software a ser desenvolvido.

Desta forma, desde pequenos detalhes como a maneira pela qual as imagens para os testes do software seriam adquiridas até decisões mais importantes como a linguagem de programação com que o software seria desenvolvido, o algoritmo que seria utilizado para a identificação de características representativas das impressões digitais e a maneira como os dados seriam distribuídos entre os diversos processadores tiveram que ser ponderados e para cada aspecto deste, chegou-se a uma conclusão relativa a que rumo seguir, lançando-se mão, para tanto, de técnicas como Matrizes de decisão e sessões de "brain storms".

Portanto, neste capítulo serão apresentados as características que foram tratadas individualmente e as alternativas para cada uma delas. Posteriormente, serão expostas as decisões tomadas e suas justificativas.

3.1 Aquisição de Imagens

O problema de como proceder a aquisição de imagens é o primeiro que deve ser encarado no desenvolvimento de um software de reconhecimento de padrões. As alternativas que foram encontradas para a solução deste problema foram:

- aquisição de imagens através do scaneamento de imagens diretamente roladas sobre papel;

- aquisição de imagens através de *scanner* apropriado para este tipo de aplicação (em geral com um pequeno slot para o dedo, garantindo a inexistência de rotações consideráveis) e
- aquisição de imagens através da conversão de imagens do tipo .wsq (Wavelet Scalar Quantization) para .bmp (Windows Bitmap) e eventualmente para outros tipos conhecidos e manipuláveis de imagens.

3.2 Linguagem de Programação

A escolha da linguagem de programação a ser utilizada para o desenvolvimento do projeto é um ponto crucial do trabalho, uma vez que grande número de dificuldades que serão inevitavelmente encontradas durante a fase de desenvolvimento será fruto direto das particularidades da linguagem escolhida.

Os critérios fundamentais na escolha da linguagem de programação para o presente projetos são: disponibilidade de ferramentas para a implementação de sistemas distribuídos, robustez de implementações (já que a aplicação é inerentemente computacionalmente complexa) e portabilidade para o uso em diferentes plataformas, presença de conceitos modernos de programação (como a orientação a objetos e abstração de implementação).

Naturalmente surgem como potenciais candidatos para linguagem de programação do presente trabalho as linguagens C, C++, Pascal e Java.

3.3 Segmentação de Imagens

A segmentação de imagens consiste em um pré-processamento que se faz necessário em certas aplicações que exigem que certa porção da imagem original seja descartada e que se identifique uma ou mais áreas cujo interior represente a porção significativa da imagem, que será considerada efetivamente na aplicação. Além de separar a região efetivamente relevante da

imagem (área onde se encontra a impressão digital), a segmentação pode apresentar algumas outras funções visando padronizar a imagem, como rotações, translações, ajustes de escala, etc.

Figura 1 - Imagem scaneada de um cartão antes da segmentação

Portanto, deve-se decidir se será ou não necessária a implementação de uma rotina de segmentação da imagem, além da extensão desta rotina (o quão abrangente ela seria, o quão genérica, o que exatamente seria necessário que ela fizesse, etc.). No caso de se chegar a conclusão de que será necessária tal rotina, deve-se também optar por se utilizar ou não uma versão de segmentação de imagem implementada em C++ também disponível no CD elaborado pelo NIST (National Institute of Standards and Technology) em conjunto com o FBI (Federal Bureau of Investigation), referente ao projeto AFIS (Automated Fingerprint Identification System).

Figura 2 - Figura anterior após um processo de segmentação

Uma outra característica importante da segmentação, dependendo da aplicação, é marcar áreas da imagem que deverão ser consideradas nos cálculos da aplicação visando a economia de tempo. Assim, em condições particulares, pode-se na segmentação determinar áreas que estão "borradas" demais para que o programa perca tempo posteriormente tentando dali extrair características importantes.

3.4 Processamento de Imagens

Um outro ponto que surge quando se está procedendo as especificações do problema concerne ao problema do processamento de imagens. Este assunto depende diretamente do algoritmo selecionado para a extração de características representativas, uma vez que alguns algoritmos exigem que a imagem seja binarizada (em preto e branco apenas), outros exigem que a imagem tenha seu contraste realçado com um filtro do tipo FFT (Fast Fourier), outros demandam uma imagem "esqueletizada" (com linhas de apenas um pixel de espessura) e assim por diante.

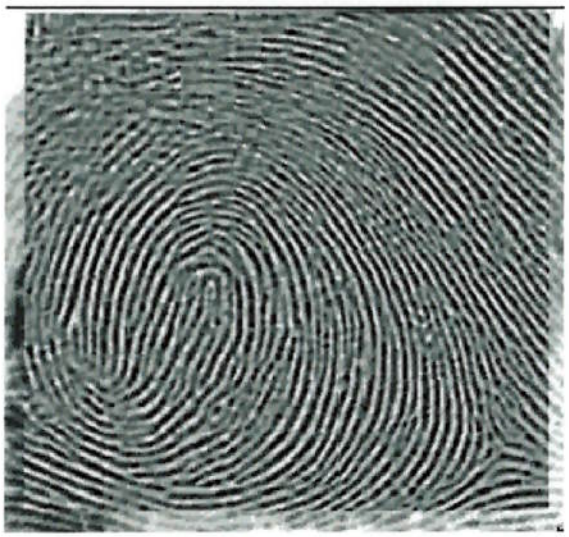


Figura 3 - Imagem anterior com contraste realçado

Assim, decidido o algoritmo que será implementado no projeto, pode-se ponderar o(s) processamento(s) de imagem necessário para atender todas as particularidades do respectivo algoritmo selecionado.

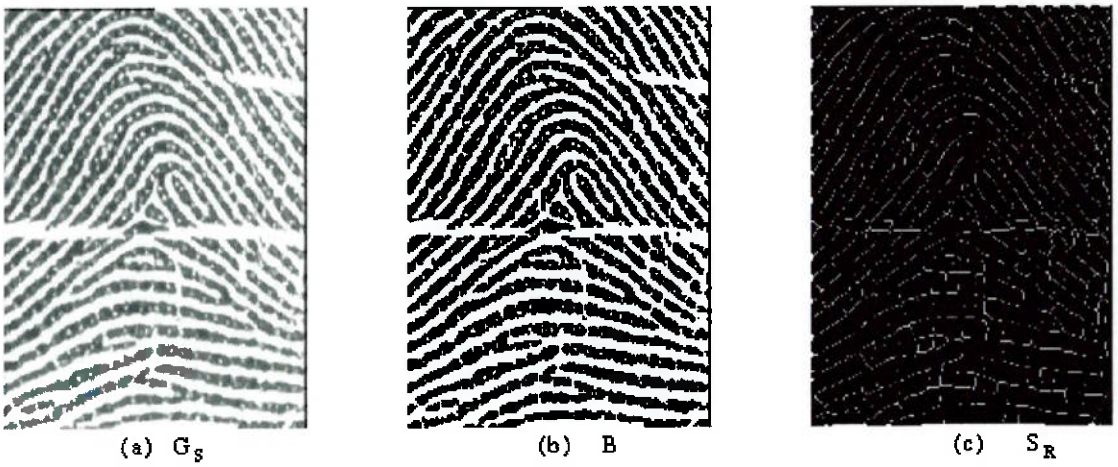


Figura 4 - Exemplos de processamentos de imagem
(a) imagem original (b) imagem binarizada (c) imagem esqueletizada

3.5 Identificação de Características Representativas de Digitais

Esta é talvez a característica do projeto mais difícil a ser definida em todo o trabalho, uma vez que impactará diretamente sobre a maneira como o processamento será distribuído em diversas máquinas. Portanto, deve-se ter bem claro os objetivos acima expostos, orientando nossa decisão não só pelas qualidades do algoritmo de extração de características representativas de digitais, mas também pensando na qualidade dos possíveis métodos de paralelização para o algoritmo escolhido, obtendo ganho de performance e sendo portável para ambientes heterogêneos.

Existem fundamentalmente dois métodos para a extração de características representativas para a posterior comparação de duas imagens de impressões digitais: extração de minúcias e identificação de orientações locais (ainda existem alguns métodos que comparam os gradientes locais, associados às minúcias ou às orientaçãoes). Minúcias são particularidades características que costumam ocorrer em impressões digitais, tais como fins de linhas ("ridge ends") e bifurcações. Localizando-se um certo número de minúcias, procede-se a comparação através de alguns métodos, sendo o mais simples deles a distância euclidiana. Já no método de orientações, obtém-se uma matriz com as orientações locais em cada pequena porção da imagem e a comparação é feita através da redução da matriz a um vetor com menos elementos através de uma transformação linear razoavelmente complicada. A seguir serão explicados com mais detalhe os dois métodos clássicos para a identificação de características representativas (observe-se que tanto minúcias quanto orientações são características representativas que podem ser usadas para a identificação pessoal).

3.5.1 Extração de Minúcias

Conforme exposto acima, uma abordagem muito usada nos algoritmos para a extração de características representativas das digitais é a extração de minúcias, que podem ou não ser feitas sobre imagens binárias, esqueletizadas ou até mesmo sobre a imagem em tons de cinza.

Para imagens esqueletizadas, pode-se identificar minúcias através do mapeamento das linhas e será considerada como bifurcação aquele pixel que tenha exatamente três vizinhos não nulos e fim de linha aquele pixel que tenha apenas um vizinho não nulo.

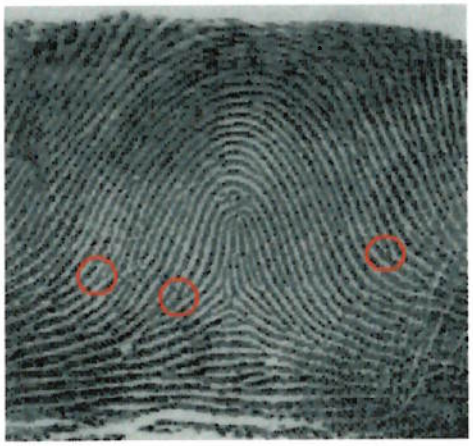


Figura 5 - Exemplos de bifurcações em impressão digital

Para imagens em tons de cinza é apropriado o uso de lógica fuzzy, que identifica minúcias através da subdivisão da imagem original em blocos menores, normalmente quadrados, onde no interior desta região considera-se uma sub-região de n x n pixels e outras oito regiões correspondentes a faixas nas bordas (norte, nordeste, leste, sudeste, sul, sudoeste, oeste, noroeste). Destas oito janelas, observa-se que apenas seis delas por teste são suficientes para a extração de minúcias. Cada faixa da borda tem a si associada um grau de cinza e de branco. Assim, testa-se para cada região a possibilidade de que ali haja uma bifurcação cuja orientação seja norte, nordeste, e assim por diante. Também testa-se se a hipótese de ali haver um

fim de linha de orientação norte, nordeste, e assim por diante. Para o teste de hipótese de haver na região uma determinada minúcia, vale a regra de que ali há esta determinada minúcia com a "força" mínima entre todas as faixas da borda, ou seja, se em uma bifurcação do tipo norte deve haver a faixa sul branca e as outras cinco faixas opostas cinza, diz-se que esta região tem uma bifurcação norte com a mínima força dentre a força com que a faixa sul é branca e as outras são cinza. Analogamente, apura-se a força de determinado tipo de minúcia na região pela regra do máximo, ou seja, se numa região, há 0,1 de bifurcação norte, 0,05 de bifurcação nordeste, e assim por diante, apura-se qual é a minúcia que tem maior força naquela região e dependendo de seu valor, apura-se se ele é suficiente para que se considere que ali há mesmo uma minúcia ou não. Identificadas as minúcias, a codificação das mesmas pode ser feita de várias maneiras. Uma maneira alternativa armazenaria a posição x, y da minúcia, o tipo (Bifurcação ou Fim de Linha), a orientação, etc. Uma forma muito usada em grandes bancos de dados de digitais é codificar apenas a posição x, y e o tipo e a codificação poderia ser feita em uma string, com a vantagem da boa compressibilidade dos dados assim codificados.

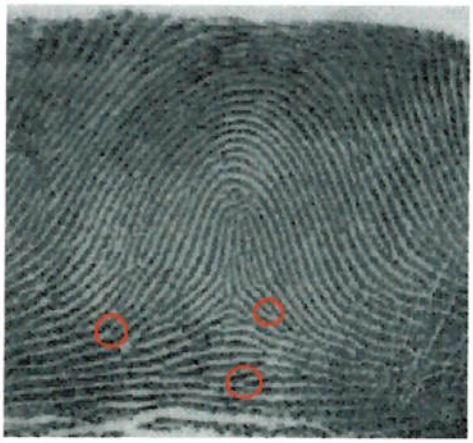


Figura 6 - Exemplos de "fins de linha" em impressão digital

3.5.2 Extração de Orientações Locais

A segunda abordagem clássica para a obtenção que características particulares de cada impressão digital é extrair orientações locais das linhas e dos vales, de forma a se obter uma matriz de médias locais destas orientações. Uma maneira simplista de obter-se orientações locais seria, numa imagem binarizada, tornar a orientação de um pixel branco igual à direção de sua mínima soma e analogamente a orientação de um pixel preto igual à direção de sua soma máxima. O grande problema deste método seria a grande suscetibilidade a ruído e além disso vetor de orientações seria excessivamente grande.

Uma alternativa razoável seria reduzir-se o vetor de orientações para um bem menor, onde cada orientação representaria a média das orientações de um bloco de 16 x 16 pixels. Cada ângulo local seria definido como 0° para uma linha horizontal, e variando até 180° conforme a linha gira em sentido anti-horário e se torna novamente 0° quando a linha se torna novamente horizontal. A média é calculada baseada nos vetores de orientação que têm como pares ordenados ($\cos 2\theta$, $\text{sen}2\theta$). Como todos estes vetores orientação tem comprimento unitário, a média tem comprimento no máximo um também o que faz com que regiões com muito ruído, borradas por exemplo, e que não tenham uma orientação muito bem definidas, apresentem um vetor médio de comprimento pequeno, já que as orientações dos 256 pixels tenderão a se cancelar. Este artifício é importante porque assim a média local dos comprimentos dos vetores próximos a cada região é uma medida da representatividade daquela parte da imagem, e blocos com comprimento pequeno indicam menos certeza naquele valor médio de orientações.

Neste caso em que as orientações são obtidas, é a partir delas é que se fará a comparação entre as digitais, é importante neste ponto resolver o problema de possíveis translações que possam ter ocorrido quando da aquisição das impressões digitais. Um algoritmo que trata de fazer este ajuste é o R92, utilizado pelo NIST no projeto do PCASYS (A Pattern-level

Classification Automation System for Fingerprints). Este algoritmo acha um ponto característico na impressão digital correspondente ao núcleo (ou ao núcleo de menor ordenada) e translada a digital colocando este ponto de registro sobre um ponto padrão obtido pela mediana dos pontos de registros de uma amostra considerável de digitais.

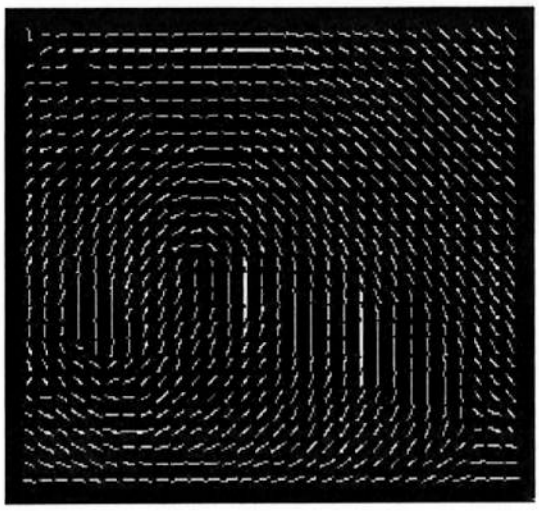


Figura 7 - Representação geométrica das orientações locais de uma impressão digital

O vetor de orientações, mesmo depois de ser transladado ainda não é apropriado para passar pelas comparações para a identificação de digitais. Sobre ele, é aplicada uma transformação linear objetivando reduzir a dimensionalidade do vetor de orientações e de certa forma fazer valer a diferença relativa entre as representatividades locais nas diferentes regiões da imagem da digital. Esta transformação linear é chamada de Transformada de Karhunen-Loève (K-L). Para produzir a matriz que implementa a Transformada K-L, o primeiro passo é obter a matriz de covariância amostral do vetor registrado original. Então, é usada uma rotina de diagonalização para produzir um subconjunto com *m* autovetores da matriz de covariância, correspondentes aos maiores autovalores (*m* arbitrário, usualmente na faixa de 100). Assim, para qualquer n menor ou igual a m, a matriz Ψ pode ser definida como tendo em suas colunas os primeiros n autovetores, e cada autovetor tendo tantos

elementos quanto o vetor de características original. Uma versão da Transformada K-L que reduz um vetor original u para um vetor w de n elementos pode ser definida como:

$$w = \Psi^{t} u.$$

A Transformada K-L, portanto, reduz o vetor de orientações para um vetor com dimensões muito menores que pode ser utilizado para efeitos comparativos com resultados muito parecidos com aqueles que seriam obtidos se a comparação fosse realizada antes da Transformada de K-L.

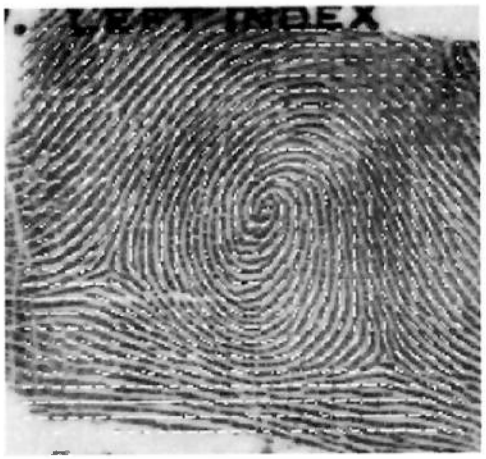


Figura 8 - Impressão digital com orientações locais indicadas

3.6 Comparação de Impressões Digitais

Naturalmente o método de comparação entre as impressões digitais depende diretamente das características representativas a serem extraídas. Assim, dependendo do algoritmo, pode-se desejar comparar vetores, minúcias e em alguns casos até mesmo a localização de poros.

O método da Distância Euclidiana é talvez o método mais simples e intuitivo de se comparar dois vetores n-dimensionais representativos das

características de impressões digitais (vetores em geral obtidos após a utilização da Transformação de Karhunen-Loève). Consiste basicamente em calcular a distância entre dois vetores x e y (n-dimensionais) como se segue:

$$D(x,y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2 + (x_3 - y_3)^2 + ... + (x_n - y_n)^2}$$

Estatisticamente (através de um grande número de testes) calcula-se um valor limiar ótimo para a distância *D(x,y)* a partir do qual considera-se que duas impressões digitais não são da mesma pessoa; caso a distância seja menor que este valor, considera-se as duas digitais como sendo da mesma pessoa.

Ainda existe um outro método utilizado para estes fins chamado de Distância Quadrática. Neste método, faz-se uso da matriz de covariância da amostra, chamada de S. Define-se esta distância quadrática como sendo:

$$D(x,y) = -(x-y)^T S^{-1}(x-y)$$

Também neste caso, x e y são vetores n-dimensionais , por T denota-se a transposição do vetor e por S^{-1} denota-se a inversa da matriz de covariância da amostra.

Diferentemente dos métodos anteriores, há ainda o caso em que a entrada é uma série de minúcias identificadas por sua posição (x , y). Tem-se uma série de impressões às quais deseja-se comparar uma dada impressão (alvo). A idéia é que cada uma das impressões da base de dados tenha um número determinado de minúcias, por exemplo 10. O método fuzzy consiste em estabelecer um limite tanto para distâncias (que visam minimizar efeitos de translações indesejadas) quanto para gradientes (assume-se que duas minúcias devam ter gradientes locais com valores parecidos). Desta forma, o que se faz é basicamente verificar se na imagem alvo, na posição equivalente a cada uma das minúcias das imagens da base de dados, existe uma minúcia que esteja deslocada de um distância suficientemente pequena para que o

grau de pertencimento tenha um valor próximo de um. Caso não haja minúcia naquela região ou a natureza da minúcia seja diferente (bifurcação x ridge end), o grau de pertencimento é determinado como zero.

Analogamente, é determinado também um grau de pertencimento baseado no gradiente local , de forma que duas minúcias que estejam deslocadas de uma distância menor que o limite também sejam comparadas também quanto ao gradiente, ou seja, quanto a variação nos tons de cinza em diferentes direções.

Finalmente estabelece-se uma relação entre os dois graus de pertencimento entre duas minúcias, que pode ser uma relação qualquer que mantenha o grau de pertencimento global entre 0 e 1, por exemplo a multiplicação dos dois valores, o menor valor entre os dois, etc. Desta forma, para cada minúcia na impressão da base, tem-se um grau de pertencimento diferente na impressão alvo. Escolhe-se também uma relação entre estes graus de pertencimento, que pode por exemplo ser a média entre os graus de pertencimento de todas as 10 minúcias, e se este média for maior que um limite determinado experimentalmente, admite-se serem iguais as duas impressões digitais. Uma outra abordagem seria estabelecer um limite para dizer se cada minúcia é ou não reconhecida como tendo uma correspondente na impressão alvo, e pode-se estabelecer um limite de n minúcias necessárias para que uma impressão seja reconhecida como idêntica à outra.

Naturalmente, pode-se ainda comparar impressões digitais aproveitando-se um pouco de cada método citado anteriormente. Pode-se, por exemplo, ter como entrada uma base de dados dando conta de um determinado número de minúcias e duas coordenadas, e fazendo-se uso da distância euclidiana comparar os vetores raio (a partir do centróide da imagem) de cada uma dela com a minúcia correspondente (quando houver) na outra imagem.

3.7 Computação Distribuída

Para que se tome uma decisão quanto a melhor maneira de distribuir as tarefas entre diferentes processadores, é necessário antes se ter um background em computação distribuída, que por ser uma tecnologia recente, nos obriga a fazer aqui algumas definições e esclarecer alguns conceitos à ela relacionados.

3.7.1 Motivação

É fácil se espantar com o avanço dos modernos computadores. Nos últimos anos, barreiras de desempenho foram sistematicamente superadas, em sucessivos lançamentos da gigantesca indústria de informática. Os computadores de hoje são capazes de realizar tarefas que superam por ordens de grandeza a complexidade das tarefas de seus similares apenas alguns anos mais antigos.

A indústria de informática tem sido capaz de fazer crescer em progressão geométrica a capacidade processamento de seus produtos, com equipamentos cada vez mais poderosos, e arquiteturas que conferem novas habilidades a suas máquinas.

Por outro lado, a indústria de software é capaz de surpreendentemente "consumir" cada avanço dos equipamentos onde rodam seus programas, seja concebendo programas que realizam novas e mais complexas tarefas, seja desenvolvendo programas que realizam as antigas tarefas com ainda mais qualidade e consequentemente exigindo mais recursos dos equipamentos utilizados.

Além da demanda por performance, uma nova realidade se apresenta nos departamentos de "informática" de ambientes corporativos. Os computadores agora estão (e estarão cada vez mais) "conectados", ligados em

redes corporativas e estas, por sua vez, em redes heterogêneas de amplo alcance (como a Internet, principalmente).

Com o desenvolvimento deste cenário, duas realidades motivaram (e continuam motivando) grupos de engenheiros e cientistas da computação na direção da computação distribuída:

- Presença de máquinas cada vez mais poderosas, com recursos suficientes para capacitá-las a aplicações de alta performance antes exclusivas de complexos (e dispendiosos) sistemas de computação.
- Ambientes inteiramente conectados, em redes cada vez mais velozes (as redes locais e suas "interfaces" de usuários, as intranets), permitindo o desenvolvimento de aplicações.

Com máquina melhores e mais rápidas, individualmente mais capazes, e agora ancoradas em redes também cada vez mais rápidas, a idéia da computação distribuída ganha cada vez mais força, e desponta como uma realidade promissora. A idéia de se realizar tarefas complexas lançando mão da disponibilidade de vários processadores (localizados eventualmente em várias máquinas) oferece às empresas que as utilizarem a possibilidade de realizar suas tarefas mais complexas em seus computadores, eventualmente durante períodos ociosos dos mesmos.

Para que a idéia, no entanto, seja aplicável, era necessário obter razoável facilidade e flexibilidade na implementação de tais sistemas. Conforme visto, tivemos o surgimento da computação paralela no ambiente acadêmico, com a presença de máquinas bastante complexas, dedicadas às tarefas do processamento paralelo e que dificilmente poderiam ser usadas em ambientes não tão controlados quanto os dos laboratórios de computação de alto desempenho das universidades e centros de pesquisa.

Um terceiro elemento surgiu, então, como uma solução simples e eficaz no sentido de se poder conceber sistemas portáteis e que pudessem usar os recursos de uma gama extremamente heterogênea de máquinas presentes,

por exemplo numa Intranet de uma grande corporação: O surgimento de sistemas independentes de plataforma, notadamente a linguagem Java.

Com o surgimento do Java, resolveu-se o sério problema que era a excessiva complexidade de se realizar tarefas distribuídas em ambientes corporativos. É fácil concluir que poderíamos juntar o grande número de máquinas presentes na Intranet de uma corporação, ligadas em rede, para tentar constituir um "grande computador" virtual, resultado das somas das capacidades de processamento de todas essas máquinas, com suficiente capacidade de processamento para realizar tarefas até então restritas a dispendiosos sistemas de supercomputação. O problema é que a complexidade de se levar a cabo este "experimento" era de tal forma alta que simplesmente tornava a possibilidade pouco interessante para a grande maioria das corporações (tanto é que, hoje ainda, não se observa nas redes de grandes corporações a computação distribuída em sua Intranet).

Com a linguagem Java, entretanto, foi possível conceber sistemas de gerenciamento extremamente simples, e ainda, concentrar os esforços de projeto e criação do sistema na programação orientada a uma máquina "virtual", que recebe em cada máquina real uma implementação, permitindo-se ignorar as diferenças entre as várias máquinas existentes na rede-alvo da computação distribuída. Além dessa vantagem de simplificação do projeto, também o gerenciamento disto pode ser incrivelmente simplificado. Colocar um desktop à disposição do grande "computador distribuído" torna-se tão simples quanto acessar, com o browser já instalado do desktop, por exemplo, uma página que distribui os recursos da computação distribuída dentro da Intranet. No ítem seguinte, faremos inicialmente uma discussão entre os dois grandes modelos de computação paralela, o modelo de memória compartilhada e o modelo distribuído. Na seqüência, apresentaremos os principais tópicos da concepção de sistemas de computação, notadamente o da balanceamento de processamento entre nós (processadores) e os efeitos de tempos de comunicação entre nós na performance e sintonia fina destes sistemas.

O conhecimento de todos estes aspectos da computação distribuída é necessário para que possamos no capítulo seguinte decidir como implementar o sistema de reconhecimento de impressões digitais da melhor maneira possível.

3.7.2 Modelos de Computação Paralela

A chamada computação paralela, conforme já exposto, surgiu inicialmente em ambientes acadêmicos (laboratório de computação de alto desempenho) e seu primeiro produto foram os computadores paralelos de memória dista compartilhada.

Nestes computadores, o paralelismo se dá por meio de uma implementação em hardware. Num computador paralelo de memória compartilhada, temos uma máquina onde a memória (RAM) é comum a todos os nós de processamento, podendo todos eles acessar a mesma, desde que seguindo regras que impedem conflitos entre os nós por uma dada página da memória.

O paralelismo é implementado via hardware. Nestas máquinas, tipicamente, temos cada instrução "distribuída" para o próximo processador livre (a não ser que o software tome para si a deliberação sobre a escolha do processador para executar uma instrução), e dessa forma, temos, na média, um significativo ganho de desempenho.

Esse modelo de computação paralela apresenta uma vantagem muito grande: Ele é, de certa, forma, "transparente" para a aplicação. Uma aplicação desenvolvida para uma máquina com memória compartilhada não precisa se "preocupar" em como usar o paralelismo da máquina, já que o próprio hardware dessa fará uso do paralelismo, distribuindo a carga entre seus processadores. A grandes desvantagem é que exatamente por termos que controlar um número muito grande de processadores "querendo" acessar uma memória única, temos uma limitação quanto ao número de processadores que

se pode ter numa máquina como essa. Quando observamos a curva de "speed up" (isto é, a curva que descreve o ganho de desempenho que se obtém do uso de mais processadores pelo número de processadores do sistema), temos que normalmente a curva para máquinas de memória compartilhada torna-se muito ruim (comparando-se com a de outros modelos) quando ultrapassamos um número considerado pequeno de processadores.

Naturalmente, não será um sistema de memória compartilhada o modelo que será utilizado no presente projeto, uma vez que a elaboração de computadores paralelos fugiria ao escopo deste trabalho.

Um segundo modelo de computação paralela é o chamado modelo de memória "distribuída". Neste modelo, ao contrário do anterior, temos a memória do sistema "distribuída" entre várias unidades de processamento (processadores), de forma que cada unidade de processamento tem sua porção de memória. O modelo distribuído permite que tenhamos um número muito grande de nós, já que não há "conflitos" entre os nós pelo acesso à memória. Sua maior desvantagem é o fato de ser necessária, uma vez que a memória não é compartilhada, muita comunicação entre os nós, já que, numa situação ideal, cada nó recebe a porção de memória pertinente à realização da subtarefa que está tentando realizar no momento. Trocas de grandes "pedaços" de memória entre os nós acabam congestionando a "rede" que os nós utilizam para comunicação.

No modelo distribuído, entretanto, exige-se ainda do desenvolvedor do software e sistema operacional uma política inteligente para lidar com problemas que limitam a performance do computador paralelo tais como: Política de distribuição de memória entre os nós, política de distribuição de processamento para os nós, política de sincronização de atividades dos nós.

Em máquinas típicas de memória distribuída, temos cada processador com seu próprio sistema operacional rodando em paralelo, configurando-se assim os modernos computadores paralelos de memória distribuída como um conjunto homogêneo de vários computadores de menor desempenho, ligados por uma rede ultra-rápida (o barramento da máquina paralela).

Além disso, temos o surgimento de sistemas de computação paralela constituídos de um certo número de máquinas "inteiras" (isto é, computadores inteiros) conectados à redes de alto desempenho rodando sistemas operacionais que fazem cada computador, individualmente, agir como um nó de uma grande máquina distribuída. Um dos bordões repetidos sempre por Scott McNeeley, CEO da Sun Microsystems (criadora da linguagem Java) é que, no futuro, a rede será o computador. É o que pretendemos experimentar neste trabalho.

3.7.3 Balanceamento de carga em sistemas distribuídos

Um dos maiores problemas no design de ambientes de computação distribuída é o desenvolvimento de técnicas eficientes para distribuir a carga total de processamento entre as várias unidades de processamento de forma a obter-se um desempenho global ótimo mesmo em face das dificuldades associadas à essa distribuição (que serão exploradas em maior profundidade em seções posteriores deste trabalho). O conceito de ótimo, neste caso, é relativo à variáveis como menor tempo global de execução, mínimo tempo de latência em comunicação entre nós, melhor utilização dos recursos computacionais (nós) possível, dentre outros. Alguns ambientes de computação distribuída implementam isso de maneira estática, outros de maneira dinâmica e a maioria adota uma solução híbrida de compromisso entre as duas alternativas anteriores, como veremos a seguir.

3.7.3.1 Balanceamento estático

Num sistema com balanceamento estático de carga, a separação de tarefas por para os diversos processadores é feita antes da execução do programa, ou seja, em tempo de compilação. Desta forma, deve-se usar os conhecimentos acerca do ambiente de execução (capacidade dos

processadores, latência e banda de comunicação entre os processadores, principalmente) e da tarefa a ser executada para ainda durante o desenvolvimento do programa a ser rodado no ambiente distribuído concebê-lo de forma a separá-lo em tarefas distribuídas entre as unidades de processamento. O objetivo final, que é reduzir o tempo total de execução e tempos de comunicação entre-nós é alcançado tentando-se predizer o comportamento do algoritmo a ser paralelizado quando de sua execução e tentando particioná-los em subtarefas tão locais quanto possível (isto é, tarefas que dependam apenas de pequenas porções do total dos dados a serem utilizados e que preferencialmente independam dos resultados da execução das demais subtarefas). Quanto mais eficiente for esta subdivisão, menor necessidade existirá de comunicação entre os nós, e melhor será o tempo de execução uma vez que teremos máximo uso das capacidades das unidades de processamento (já que minimizaremos as paradas de algumas unidade à espera de dados ainda incompletos de outras unidades).

3.7.3.2 Balanceamento Dinâmico

Neste tipo de balanceamento, o mesmo ocorre durante a execução do programa. Isto significa que, no design do programa (tempo de compilação), projetou-se o mesmo de forma que este seja capaz de, dependendo das condições do ambiente de processamento, distribuir tarefas entre os nós de modo a reduzir o tempo total de processamento.

Para tanto, o sistema deverá ser capaz de "perceber" quando algum nó está sobrecarregado com seu processamento e ser capaz de, neste caso, particionar a tarefa que o mesmo está tentando realizar em subtarefas redistribuídas entre os processadores, sempre levando em conta os requisitos desta subdivisão (mínimos tempos de comunicação e máxima localidade do processamento).

Na maioria dos sistemas com distribuição dinâmica, temos a tarefa de gerenciar essa distribuição atribuída a um único processador (que pode,

entretanto, realizar também outras tarefas). Essa centralização é útil porque centralizando-se a mesma num único processador é fácil "ensinar" a esse processador parâmetros muito relevantes na tarefa de distribuição, como a capacidade de cada nó, sua eventual especialização (a maioria das máquinas paralelas distribuídas, por exemplo, têm nós "especialistas" em i/o, que são os responsáveis pela comunicação dos demais, o que significa que tarefas i/o intensivas devem, numa distribuição ótima, ficar nestes nós e não em quaisquer outros), as condições de comunicação às quais cada nó está sujeito (em ambientes heterogêneos, podemos possuir nós cuja capacidade de comunicação é inferior ou superior, devido a uma maior ou menor banda de comunicação ou latência).

3.7.3.3 Balanceamento híbrido

Os sistemas de balanceamento dinâmicos apresentam evidentes vantagens frente aos sistemas estáticos. Eles são muito mais flexíveis e tolerantes às variações no ambiente de computação distribuída. Além disso, eles permitem o uso em sistemas não controlados. Imaginemos, por exemplo, que desejemos executar um processamento num ambiente que é também utilizado para várias outras tarefas. Como existe um elemento "perturbador" no sistema (a existência de outro processamento sobre o qual não temos qualquer informação e que potencialmente pode estar sobrecarregando alguns nós), a capacidade de o sistema de computação distribuída poder "detectar" a carga no processador (e portanto, sua "disponibilidade") para a realização de tarefas é desejável e permite um melhor uso dos recursos disponíveis.

Por outro lado, a especialização e o conhecimento prévio tanto da tarefa quanto do ambiente de computação disponível pode levar o designer do sistema inserir neste regras "estáticas" mas eficientes de distribuição. Por exemplo, imaginemos um ambiente com vários processadores idênticos, e uma tarefa que pode ser subdividida em subtarefas idênticas e independentes. Fica claro que a distribuição ideal é a subdivisão da tarefa em tantos processadores

quanto existirem e atribuição de tarefas igualmente "pesadas" para cada um dos processadores. Um ambiente dinâmico provavelmente atribuiria toda a tarefa para um processador, que ficaria sobrecarregado e repassaria parte da tarefa para outros, e dessa forma a tarefa seria distribuída. Obviamente, se o algoritmo de distribuição das tarefas for realmente inteligente, com o tempo, a distribuição de tarefas tenderá para a ideal. Entretanto, um tempo potencialmente precioso terá sido perdido até que cheguemos a esse ponto.

O ideal, então, é uma solução de compromisso entre os dois tipos de balanceamento, e dependerá da aplicação e das condições do ambiente de computação disponível para a aplicação desejada. Deve-se usar todo o conhecimento prévio possível para tentar gerar um sistema de distribuição inteligente atentando-se, entretanto, para as prováveis necessidades de flexibilidade que o ambiente de computação distribuída exige.

3.7.4 Granularidade

Conforme visto, a eficiência da execução de uma tarefa com processamento paralelo depende da forma como particionamos o programa em módulos de execução e os atribuímos às unidades de processamento. Os fatores que normalmente influenciam negativamente na performance do sistema como um todo são:

- **Overhead de comunicação:** Isto é, as perdas de tempo hábil de processamento quando um processador espera pela chegada de dados relevantes para a execução do processamento a ele atribuído.

- **Overhead de sincronização:** Isto é, as perdas de tempo hábil de processamento associadas à necessidade de um dado processador esperar pela execução de tarefas ainda em andamento nos demais processadores, de cujas saídas o processador em questão depende.

- **Perda de eficiência com a saída de processadores:** Isto é, a perda de eficiência (e consequentes atrasos no processamento) quando um nó deixa de estar disponível (por qualquer motivo) para realizar as tarefas de processamento desejadas.

- **Overhead de gerenciamento do paralelismo:** Além dos tempos anteriores, um sistema com processamento distribuído tem que compensar, com separação das tarefas, o tempo que ele próprio gasta com o gerenciamento dos vários processadores e com os algoritmos executados para separar as tarefas, sincronizar nós e etc.

Chamamos de granularidade das subtarefas criadas pelo sistema de gerenciamento e distribuição de carga computacional a relação entre o tempo de computação desta subtarefa e o overhead de comunicação (tempo de comunicação) a ela associada. Desta forma, um bom sistema de distribuição de carga deve objetivar a geração de tarefas com alta granularidade (o que significará que o tempo que se perde com a comunicação necessária à obtenção dos dados dos quais depende o processamento é pequeno comparado ao tempo que inevitavelmente se perderia com a execução do processamento propriamente dito).

3.7.5 Tempo de vida útil de um nó e balanceamento de carga

Conforme visto anteriormente, é desejável que o sistema de computação distribuída seja tolerante à falhas nos seus nós, isto é, que ele seja capaz de "regenerar" o processamento caso um nó tenha sofrido uma falha drástica (por exemplo, uma ausência de comunicação com o nó central por mais do que um certo tempo, por qualquer motivo).

Existem várias formas, dependentes da aplicação específica, de se regenerar o processamento que um nó começou a fazer mais não acabou porque "saiu do ar". A forma mais usual, entretanto, é simplesmente repetir-se o processamento num outro nó.

Uma vez que a falha pode acontecer, temos uma regra que contrapõe o critério da granularidade acerca da distribuição do processamento. Conforme foi exposto, o critério da granularidade nos diz que o processamento deveria ser dividido em pedaços com alta granularidade, isto é, com muito tempo computacional relativamente ao tempo de overhead de comunicação. Uma tendência natural seria, portanto, tentarmos aumentar ao máximo o tempo de computação de cada subtarefa (desde que não aumentássemos demais o tempo de comunicação para realizá-la).

Com a introdução da possibilidade de um nó falhar, entretanto, e sabendo que caso a falha ocorra o que faremos será repetir o processamento em outro nó, temos uma nova limitação na distribuição de tarefas. Deveremos dividir o processamento em subtarefas com alta granularidade sim, entretanto, sem que o tempo de computação ultrapasse um valor que faça com que na maioria dos casos os nós "saiam do ar" antes de acabar o processamento. Matematicamente, temos:

Sendo T o tempo médio que um nó permanece disponível para o processamento, sendo C_{trab} o trabalho (computacional) que uma tarefa envolve e T_{trab} o tempo que o nó leva para realizá-la, e T_{rede} o tempo de overhead de rede para esta tarefa, temos que o trabalho total que um nó realiza num período T vale:

$$C_{total} = \frac{T}{(T_{trab} + T_{rede})} C_{trab}$$

Dessa forma, percebemos que é interessante crescermos C_{trab} comparando-o com T_{rede} (critério da granularidade), mas não podemos crescer indefinidamente C_{trab} pois quando o fazemos, crescemos também T_{rede} e

corremos o risco de que quanto mais esta se aproxima de T, com maior frequência o nó vai sair do ar sem completar a tarefa por inteiro (e além disso, o "prejuízo" deste evento será maior porque o trabalho perdido também será maior).

4. Possíveis Soluções

Dadas as especificações apresentadas no capítulo anterior, serão apresentadas neste capítulo as ponderações feitas a respeito de cada característica do projeto, bem como as decisões que foram tomadas a respeito de cada uma isoladamente. No capítulo seguinte far-se-á um resumo das decisões aqui apresentadas para então apresentarmos o projeto final implementado.

4.1 Aquisição de Imagens

Dentre as três alternativas apresentadas para este ítem, talvez a que despertasse maior interesse a priori fosse aquela em que as imagens seriam obtidas a partir de *scanners* apropriados para este tipo de aplicação. Seria uma maneira prática e estimulante de proceder os testes, uma vez que as impressões digitais processadas poderiam ser dos próprios autores e de outras pessoas que estivessem presentes durante as realizações dos testes. A grande desvantagem desta alternativa era seu custo: a grande maioria dos aparelhos como este disponíveis no mercado não oferecem uma interface simples para a obtenção de imagens facilmente manipuláveis. É possível encontrar no mercado *scanners* de digitais relativamente baratos (por volta de US$ 150,00), que porém não permitem a obtenção de imagens, já que em geral estes scanners vêm acompanhado de softwares para aplicações específicas (muitas vezes, incluindo login em redes). Para a aquisição de imagens, estes aparelhos teriam de ser acompanhados por "Developers Kits" cujo custo em geral chegava a alguns milhares de dólares. Considerando todas estas dificuldades ligadas a esta alternativa, chegou-se a conclusão que a decisão deveria ficar entre as outras duas opções restantes.

Naturalmente a alternativa de scanear imagens diretamente roladas sobre papel teria um grande inconveniente que seria de efetivamente ter que conseguir com que muitas pessoas sujassem o dedo em tinta apropriada, além da falta de praticidade que este método inerentemente traria. Apesar disso, optou-se por este método de aquisição de imagens, já que assim seria possível a realização de testes mais palpáveis, com as impressões digitais das pessoas envolvidas no projeto. A desvantagem do banco de imagens do NIST consistia no fato de que a qualidade das imagens era deplorável, uma vez que as mesmas haviam sido obtidas através de impressões digitais roladas sobre papel sem esta finalidade, o que ocasionou uma grande parcela das imagens consideravelmente borradas.

4.2 Linguagem de Programação

A questão da linguagem de programação a ser escolhida para a elaboração do sistema aqui proposto é talvez a questão mais direta e fácil de ser resolvida. Por apresentar características altamente indicadas para a linguagem de desenvolvimento do projeto, como utilizar o paradigma da programação orientada a objetos, possuir ferramentas padronizadas para interface gráfica (favorecendo a portabilidade do programa) e ampla disponibilidade de recursos que facilitam a implementação de sistemas distribuídos, além da vantagem de que o byte-code é independente da plataforma (dependendo apenas de um interpretador específico à plataforma), selecionou-se a linguagem Java para ser utilizada no desenvolvimento do software proposto. A única desvantagem desta escolha reside no fato de que os programas elaborados nesta linguagem costumam ser mais lentos que programas equivalentes, plataforma-específicos, elaborados em linguagem C ou C++ por exemplo, pois o byte-code em Java é interpretado. Ainda sim, dadas as vantagens desta linguagem, parece imediato que a mesma é a mais indicada para ser utilizado neste projeto. Além disso, pela pressão resultante

de sua imensa popularização, são cada vez mais comuns interpretadores Java rigorosamente otimizados, que aceleram drasticamente o processamento de programas Java em alguma plataforma, minimizando as perdas de desempenho derivadas da interpretação do byte-code.

4.3 Segmentação de Imagens

Levando-se em conta o modo de aquisição de imagens selecionado entre as possibilidades anteriormente propostas, chega-se à seguinte conclusão: realmente será necessário um procedimento de segmentação de imagens, já que as imagens disponíveis no CD criado pelo NIST são imagens scaneadas de imagens roladas a tinta em cartões de papel, não havendo nada que garanta uma certa centralização das impressões digitais nas imagens .wsq (convertidas para .bmp e posteriormente para .gif ou .jpg). Além disso, em cada imagem existem palavras impressas correspondentes ao dedo da mão correspondente à respectiva imagem (thumb, index, ring, little, middle), que devem ser eliminadas por não serem úteis na extração de características representativas.

Poder-se-ia adaptar o procedimento de segmentação de imagens utilizado no anteriormente já citado projeto AFIS (NIST / FBI), implementado em C++, porém optou-se por elaborar um novo procedimento de segmentação de imagens em Java, uma vez que neste procedimento já poderiam ser embutidos alguns cálculos locais utilizados em outros escopos do sistema (conforme explicado mais adiante).

4.4 Identificação de Características Representativas de Digitais

Conforme já foi enfatizado anteriormente, o algoritmo que identificará um certo número de características representativas de impressões digitais é uma aplicação exemplo para o Sistema de Computação Distribuída aqui

proposto. Assim, deseja-se encontrar o algoritmo que tenha maior sinergia e adaptabilidade para o sistema distribuído em diversas unidades de processamento e naturalmente deseja-se que o algoritmo escolhido possa ser desmembrado em um grande número de tarefas relativamente independentes e computacionalmente custosas.

A primeira grande desvantagem que apresenta o algoritmo baseado nas orientações locais é a exigência por parte deste de pesados cálculos durante o pré-processamento da imagem, uma vez que este algoritmo seria muito prejudicado caso a imagem não tivesse passado por um filtro de realce de contraste. Estes cálculos são fundamentalmente globais e não apresentariam grande ganho de performance se executados em diferentes unidades de processamento. Além disso, a Transformação de Karhunen-Loève exige que se calcule uma matriz de covariância baseada num grande número de amostras, o que foge do escopo do presente projeto, já que não se trata de um classificador (como o PCASYS) e sim um extrator de características representativas. Neste sentido o algoritmo fuzzy se destaca por trabalhar na imagem original, uma vez que em cada porção da imagem se trabalha com médias locais, possibilitando a distribuição em diferentes processadores de maneira razoavelmente independente. Além disso, o processamento de cada imagem independe de um grande número de amostras; na verdade o resultado de cada imagem depende única e exclusivamente dela própria, o que significa que cada processador poderá realizar todo o processamento sem, em nenhum momento, depender de dados que não "estejam" nele localizados (não serão necessários "page faults" de memória, e portanto os requisitos de sincronia entre os processadores são mínimos).

Um outro fator importante a favor do algoritmo fuzzy é a exigência computacional deste algoritmo, que é maior (sem considerar os processamentos de imagens dos outros métodos, que não são distribuíveis ou pelo menos, não são recomendáveis à distribuição) que os outros apresentados anteriormente, uma vez que cada porção da imagem é tratada muitas vezes (pois para dois blocos n x n consecutivos considerados, há uma

superposição de pixels, evitando que uma minúcia que se encontrasse exatamente entre dois blocos consecutivos sem superposição passasse despercebida).

Desta forma, a aplicação exemplo para nosso Sistema de Computação Distribuída consistirá num algoritmo que faz uso da lógica fuzzy para identificar minúcias em imagens de impressões digitais.

4.5 Processamento de Imagens

Baseado no algoritmo selecionado para a extração de características representativas de impressões digitais (utilizando fuzzy logic para a identificação de minúcias nas impressões), elimina-se a necessidade da implementação de quaisquer métodos de binarização de imagem, e mesmo de qualquer tipo de realce de contraste, já que inerentemente o algoritmo fuzzy já tem como parâmetro as médias locais (conforme explicado anteriormente). Desta forma, também não será necessária a implementação de um filtro do tipo FFT que seria exigido caso o algoritmo selecionado tivesse sido aquele que se baseia nas orientações locais, nem qualquer esqueletização da imagem, o que seria necessário caso o algoritmo escolhido tivesse sido o mapeamento de minúcias através das linhas de 1 pixel de largura.

4.6 Comparação de Impressões Digitais

A escolha do método fuzzy como algoritmo do presente projeto descarta algumas hipóteses de comparações entre as características representativas de impressões digitais, principalmente aqueles que se baseavam na comparação de matrizes decorrentes do método de extração de orientações locais e posterior Transformação de Karhunen-Loève.

O algoritmo selecionado proporcionará como saída da etapa de identificação de características uma série de minúcias, suas coordenadas, sua

natureza e sua orientação. Como optou-se por adquirir as imagens fundamentalmente através do grande banco de dados disponível no CD do projeto PCASYS, e não foi possível utilizar um scanner próprio para este tipo de aplicação, optou-se por dar às orientações das minúcias uma importância secundária, já que não haveria como garantir que não houvesse pequenas rotações relativas entre as imagens correspondentes à mesma impressão digital que poderiam comprometer uma eventual comparação dando grande importância às orientações das minúcias.

Assim sendo, optou-se apenas por comparar as imagens de impressões digitais procurando correspondência de minúcias nas imagem consideradas. Dada uma digital já processada, testar-se-á se na imagem candidata há uma minúcia em posição equivalente (relativamente ao centróide), naturalmente dentro de uma tolerância a ser determinada. Também encontrar-se-á experimentalmente um número mínimo de minúcias que devem ser correspondidas para que o sistema considere que duas imagens são efetivamente das impressões digitais de uma mesma pessoa.

4.7 Computação Distribuída

A escolha pelo sistema de computação distribuída foi, em parte, realizada com a simples adoção da linguagem Java como a linguagem de programação.

Isto porque as versões mais novas da linguagem (e que foram utilizadas nesse projeto, a 1.1.7) possuem uma especificação de API para paralelismo bem definida e estudada com os principais desenvolvedores de soluções de computação paralela.

Essa API, chamada RMI (Remote Method Invocation), especifica como sistemas de computação distribuída baseados em Java deverão se comportar, e um estudo detalhado da mesma foi proveitoso para entender a forma de computação distribuída que a linguagem oferecia, e portanto, adaptar nosso

projeto para não fugir ao RMI, já que esse é um padrão "*de facto*" em aplicações distribuídas em linguagem Java.

Dessa forma, a computação distribuída ficou baseada num sistema de objetos "client-side" e "server-side" que interagem por meio de métodos remotos, isto é, de métodos do objeto-servidor que são "exportados" e ficam à disposição do objeto-client, isto é, o objeto-client pode chamar remotamente o método do objeto-server, e vice versa.

Essas transações (entre objetos de máquinas distintas) são gerenciadas por uma camada que intermedia as chamadas de métodos remotos e os detalhes de implementação dos mesmos (comunicação tcp/ip entre as máquinas, etc), chamado RMIRegistry (heredeiro funcional das antigas RPC – Remote Procedure Calls).

Por meio do RMIRegistry, portanto, é possível não só compartilhar métodos com máquinas remotas, bem como receber e passar parâmetros que são objetos, desde que estes cumpram características definidas pela interface Java.io.Serializable. Quando assim o são, tais objetos podem ser passados como argumentos de funções remotas, ou ainda, podem ser o retorno das mesmas, permitindo o design de objetos-processos no ambiente "server-side" e posterior exportação do mesmo para as máquinas-client que o executarão.

O sistema, então, fica composto de um elemento servidor, e um número arbitrário de clients. O servidor, através de métodos remotos do client, informa os clients de suas tarefas, eventualmente passando objetos completos para estes, contendo o processamento a ser realizado, e o mesmo é realizado do lado dos clients.

O design e gerenciamento do sistema de distribuição de tarefas e controle de sua realização nos vários processadores fica inteiramente confinado ao lado do servidor no fluxograma de execução.

Para o balanceamento de carga, foi adotado um balanceamento híbrido, em que tarefas são criadas pelo servidor e passadas aos clients, que então as executam e dinamicamente chamam um método no servidor responsável por

processar eventos que os processadores geram (como a identificação de uma dada minúcia, de um dado tipo, numa dada posição da imagem original).

A granularidade das tarefas distribuídas pelos processadores paralelos foi alterada no decurso do projeto, para grande o suficiente para evitar excessivos overhead de comunicação, e pequenas o suficiente para não permitir que diferenças anormais da capacidade de processamento de cada nó pudessem prejudicar muito o desempenho final do sistema.

5. Especificações Finais do Projeto

No capítulo 3, enunciamos as questões que deveriam ser solucionadas para que pudéssemos proceder a implementação do software, separando várias características de maneira tão independente quanto possível e sugerindo várias alternativas para a solução de cada problema. No capítulo seguinte, ponderou-se as vantagens e desvantagens de cada uma das alternativas para cada característica do projeto, chegando-se às conclusões de quais seriam as melhores soluções para cada aspecto do presente trabalho científico, sem nunca perder de vista nossos objetivos. Neste capítulo, sintetizaremos como será o conjunto completo do software e especificaremos em maior detalhe cada parte dos algoritmos a serem implementados.

O princípio do algoritmo fuzzy para extração de minúcias é o seguinte:

1. Divide-se a imagem já segmentada em vários blocos;
2. Toma-se separadamente cada bloco da imagem da impressão digital (de tamanho arbitrário; fizemos testes para 16 x 16, 12 x 12, 9 x 9 , 8 x 8) que tenha sobrevivido à segmentação e que ainda tenha apenas blocos vizinhos que também tenham sobrevivido à segmentação, o que já é uma condição de borda;
3. Avalia-se a média dos pixels para aquele bloco (o tamanho de bloco decidico foi de 12x 12);
4. Divide-se o bloco em 8 sub-blocos (norte, sul, leste, oeste, nordeste, noroeste, sudeste, sudoeste) conforme a figura seguinte;

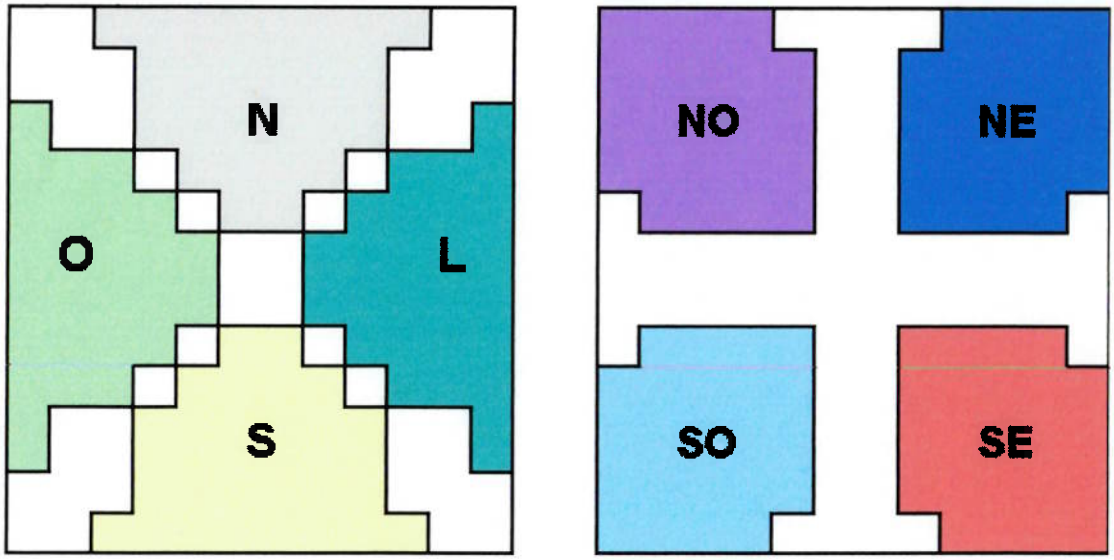


Figura 9 - Bloco 12 x 12 e sub-janelas

5. Avalia-se a média local de cada um dos sub-blocos;
6. De posse das médias locais e da média global do bloco, aplica-se uma função fuzzy que atribui a cada sub-bloco um valor de *dark* e *bright* (cuja soma é 1);
7. Com base nos valores das propriedades *dark* e *bright* de cada sub-bloco, atribui-se a ele um valor associado a cada tipo de minúcia e sua orientação (ou seja, existem 16 tipos possíveis de minúcias: bifurcações (com 8 orientações possíveis) e fins de linha (com 8 orientações possíveis)). Este valor é calculado baseado na regra do mínimo (para um determinado tipo de minúcia bifurcação Norte , por exemplo, que exige que os sub-blocos O, SO, S, SE, E sejam *dark* e o sub-bloco N seja *bright*, associa-se a este sub-bloco uma propriedade Bif.Norte cujo valor é o mínimo das propriedades *dark* dos sub-blocos O, SO, S, SE, E e da propriedade *bright* do sub-bloco N. Isto significa que se uma das janelas consideradas não se enquadrar naquele tipo de minúcia (Bif.Norte, no caso) tendo um valor próximo de zero para a propriedade considerada, o sub-bloco terá um valor baixo para aquele tipo de minúcia, sendo posteriormente avaliado que não ocorre este tipo de minúcia ali. A seguir estão alguns exemplos de minúcias e como elas se enquadram no algoritmo descrito.

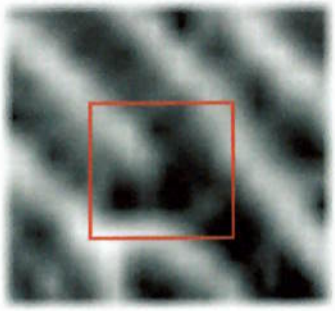


Figura 10 - Bifurcação do tipo NE

O valor associado à minúcia Bif_NE da figura anterior é dado pelo mínimo entre as propriedades dark das sub-janelas SO, S, SE, E, NE e da propriedade dark da sub-janela NO.

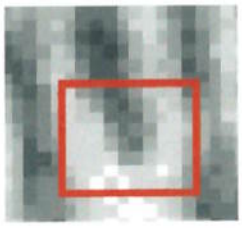


Figura 11 - Ridge End do tipo S

O valor associado à minúcia Ridge_S da figura anterior é dado pelo mínimo entre as propriedades dark da sub-janela N e das propriedade bright das sub-janelas E, SE, S, O, SO.

Tendo, para cada sub-bloco, sido avaliados os valores para cada tipo de minúcia, toma-se o valor máximo para um dos tipos de minúcia e compara-se com um valor mínimo de referência (empírico). Caso este valor seja maior que a referência, considera-se que o bloco contém uma minúcia do respectivo tipo.

Repete-se o procedimento para o próximo bloco. Observe-se que o próximo bloco não é o bloco n x n imediatamente seguinte ao último bloco analisado e sim o bloco formado pelos n x n pontos cujo primeiro ponto está

imediatamente à direita do primeiro ponto do bloco anterior. Em suma, transladou-se apenas 1 coluna para a direita, e todos os outros (n-1) x n pixels são os mesmos do bloco anterior, ocorrendo uma superposição, e cada pixel (quando não se encontra na borda) faz parte de n^2 blocos analisados. Isso gera duas consequências:

- Uma mesma minúcia pode ser identificada em vários blocos n x n próximos, ou seja, deve-se considerar como minúcia válida apenas uma minúcia dentro de uma região ao redor de um bloco n x n;
- Esta superposição evita que minúcias que porventura estivessem exatamente na divisa entre dois blocos n x n consecutivos não fossem identificadas.
- Um bloco 12 x 12 é analisado 144 vezes (cada pixel do bloco é considerado uma vez como sendo o canto superior esquerdo de um bloco 12 x 12).

Este critério de se avaliar todos os blocos 12 x 12 possíveis é um fator regulador de sensibilidade e desempenho do sistema, isto é, pode-se chegar a conclusão, empiricamente que é suficiente se avaliar um número menor de pixels como sendo o canto superior esquerdo de um bloco. Naturalmente o impacto desta decisão é quadrática no número total de blocos avaliados. O número de blocos derivados (n) de um bloco 12 x 12, com um passo p é dado por:

$$n = \left(\frac{12}{p}\right)^2$$

A seguir é mostrado o gráfico que expressa as funções Fuzzy_Bright e Fuzzy_Dark:

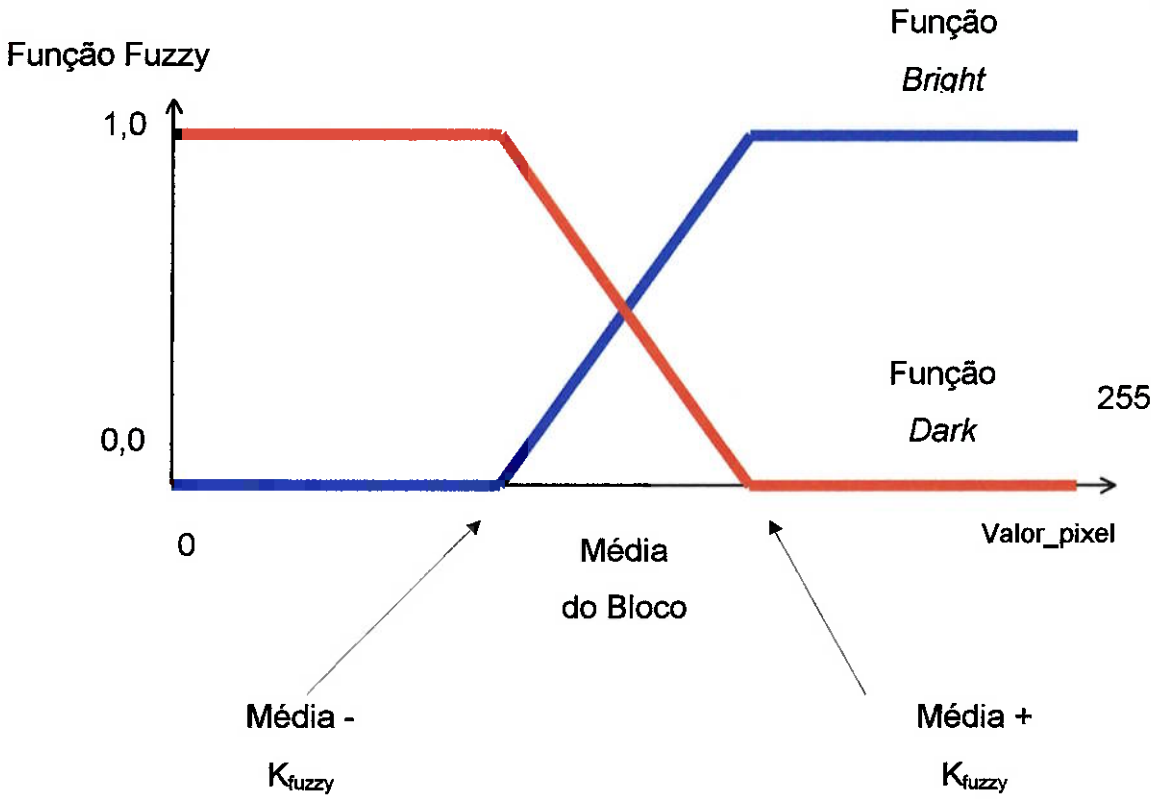


Figura 12 - Funções Fuzzy Dark e Bright, aplicadas a cada sub-bloco

Observa-se na figura acima um gráfico explicativo das funções Fuzzy *Bright* e *Dark*. O princípio das funções é a seguinte: se um sub-bloco tem sua média local muito mais escura que a média do bloco, terá sua propriedade *dark* com valor 1 e *bright* com valor 0 (parte esquerda do gráfico), e a recíproca também é verdadeira. Porém, se o valor da média do sub-bloco for próximo à média do bloco inteiro, a função fuzzy atribuirá um valor intermediário entre 0 e 1 para as propriedades daquele sub-bloco. Para tanto, é necessário que se escolha arbitrariamente um K_{fuzzy} (que foi determinado após muitos testes como 25) cujo valor implicará numa maior ou menor restritividade ao atribuir as propriedades *bright* e *dark* aos sub-blocos.

6. Engenharia do Software

Neste capítulo apresentar-se-á a estrutura do software final implementado. Primeiramente serão expostos cada classe, procedimento, função do software e ao final do capítulo uma síntese do funcionamento global do projeto será apresentada.

6.1 Classe Display_gráfico

A Classe Display_gráfico é a classe responsável por toda a interface gráfica do software no servidor. Além disso, apresenta procedimentos como o "segmenta", que é responsável pela segmentação da imagem.

6.1.1 Segmenta ( )

Responsável pela segmentação da imagem, este procedimento funciona da seguinte forma:

- Busca uma imagem (832 , 768) ;
- Cria um array blocos8x8 de [104][96] blocos 8 x 8 pixels (objeto Bloco8x8[][] explicado adiante) ;
- Atribui valores às propriedades x e y de cada elemento i, j do array blocos8x8[i][j] ;
- Cria um array de inteiros subbloco[8*8] ;
- Copia os pixels da imagem original para o array subbloco[i] ;
- Aplica o método set_pixels em cada elemento [i][j] do array blocos8x8 passando como parâmetro o array subbloco (copiando os pixels da imagem original para cada bloco8x8) e já calculando no bloco a média local, o mínimo e o máximo) ;

- Calcula a média global da imagem original ;
- Estabelece um critério inicial para atribuir a propriedade "sobrevivi" de cada bloco8x8 como a comparação com a média global (caso a média local do bloco8x8 seja mais escura que a média global da imagem, sobrevivi é verdadeiro; caso contrário, é falso) ;
- Passado o critério inicial acima, a imagem não está contínua; existem blocos 8 x 8 no interior da impressão digital que foram "eliminados" por sua alta luminosidade. Procede-se então a continuização da imagem, visando evitar "furos" na impressão digital;
- Percorre a imagem em cada linha. Caso o bloco8x8 da posição i tenha sobrevivido, o da posição i+1 não tenha sobrevivido, e o da posição i+2 tenha sobrevivido, constatou-se um "furo"; atribui-se ao bloco i+1 sobrevivi = true;
- Repete-se o mesmo procedimento acima para casos de "furos" de tamanho de 2 blocos 8 x 8, isto é, se os blocos i e i+3 sobreviveram e os blocos i+1 e i+2 não sobreviveram, atribui-se aos blocos i+1 e i+2 sobrevivi=true;
- O mesmo raciocínio apresentado acima é repetido, porém percorre-se a imagem por colunas. Retira-se furos de 1 ou 2 blocos 8 x 8 pixels;
- Até aqui garantiu-se apenas que a imagem tivesse porções contínuas porém ainda não se procedeu uma erosão propriamente dita, eliminando-se partes contínuas que não fizessem parte da impressão digital;
- A erosão horizontal é feita da seguinte forma: percorre-se cada linha da imagem. Quando se encontra um bloco que sobreviveu, começa-se a contar o tamanho desta região, e guarda-se a posição do primeiro bloco desta região válida. Quando, ao final desta região se encontra um bloco que não sobreviveu, testa-se se o tamanho desta região percorrida é o maior tamanho de regiões válidas naquela linha. Quando se chega ao final da linha, tem-se a posição do bloco que precede a maior região de blocos 8 x 8 que sobreviveram e o tamanho da respectiva região. Assim, faz-se com que todos os outros blocos que não pertencem a esta região não sobreviverem. Resumindo, acha-se a maior seqüência contínua de blocos 8 x 8 que

sobreviveram em cada linha, e faz-se com que todos os outros blocos não sobrevivam ;

- Atualiza-se a tela, repete-se o procedimento acima para cada coluna (erosão vertical), e atualiza-se a tela novamente ;
- Computa-se a posição do centróide através do cálculo das médias das posições dos blocos 8 x 8 que sobreviveram à segmentação ;
- Manda mensagem para a interface de que a segmentação está finalizada.

Abaixo, temos um exemplo de digital segmentada, com a respectiva interface "server-side" do sistema:

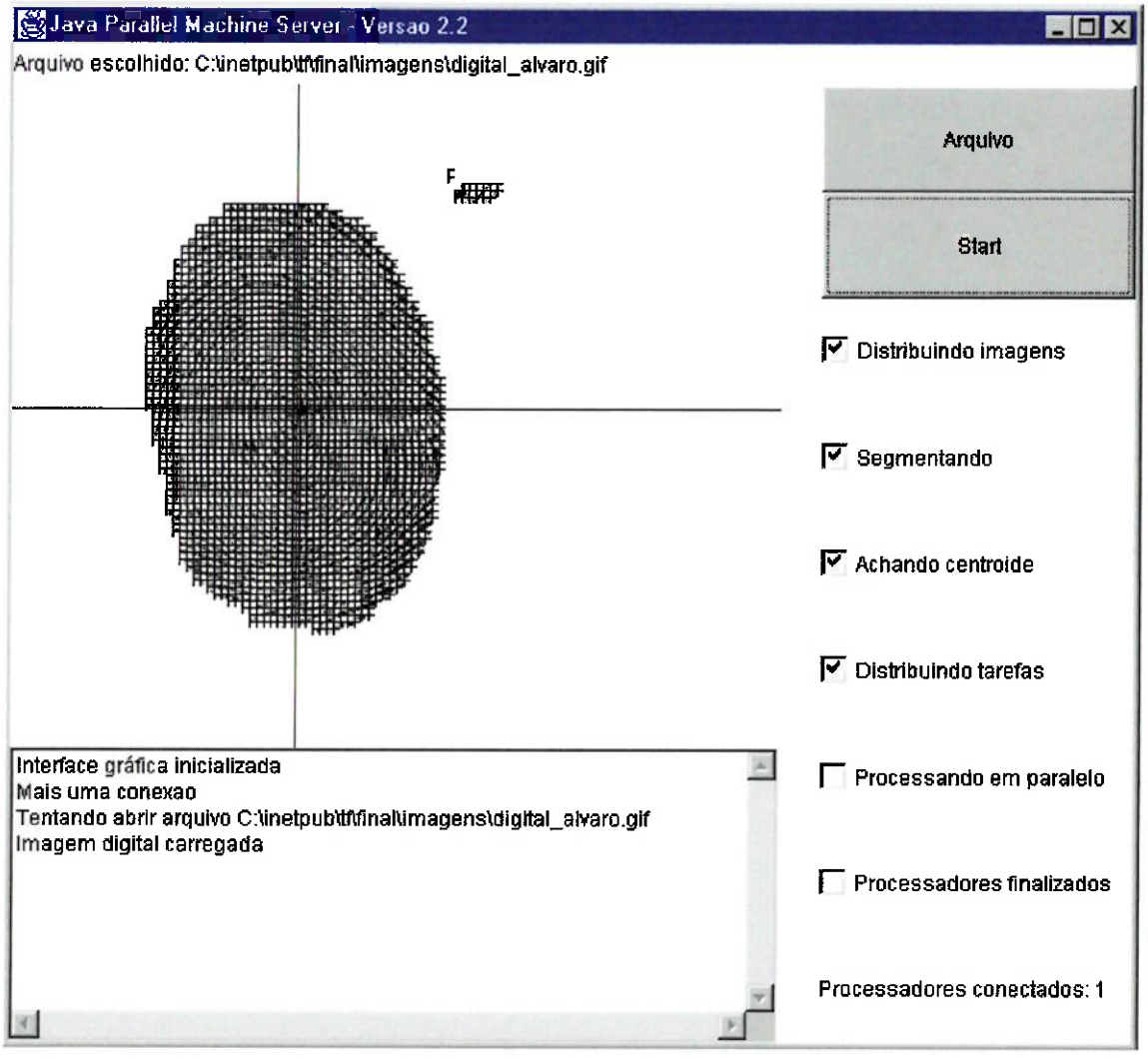


Figura 13 – Interface com imagem segmentada

6.1.2 Classe Bloco8x8

Esta classe define um bloco de 8 x 8 pixels em um vetor de inteiros, sua posição dada por um par de inteiros (x, y), e possui ainda as propriedades max_value (valor do pixel mais escuro do bloco), min_value (valor do pixel mais claro do bloco) e med_value (valor médio dos pixels do bloco).

O método set_pixels (entrada[]) recebe um vetor de 64 inteiros e os atribui ao vetor de inteiros do Bloco8x8, além de calcular a média dos pixels e acertar o valor de med_value.

6.2 Classe Bloco_Minúcia

A classe Bloco_Minúcia é responsável pela identificação propriamente dita das minúcias nas impressões digitais através do uso de fuzzy logic. Para estimar o grau de pertencimento de cada pequeno bloco da imagem aos conjuntos de cada tipo de minúcia, subdivide-se o bloco considerado (de 12 x 12 pixels) em oito sub-janelas e avalia-se as médias locais em cada uma delas, comparando-se com a média em todo o bloco considerado. Assim, à cada sub-janela atribui-se um valor dark e bright (cuja soma é um) e para cada tipo de minúcia (8 orientações de bifurcação e ridge end), usa-se a regra do mínimo (um bloco 12 x 12 tem um grau de pertencimento ao conjunto de minúcias "x" determinado pelo mínimo entre as propriedades dark e bright de suas sub-janelas nas respectivas posições daquele conjunto de minúcias).

A classe Bloco_Minúcia é estruturada da seguinte forma:

Propriedades e Variáveis

- med_value - Valor médio do Bloco 12 x 12 ;
- x, y - Coordenadas do pixel superior esquerdo do bloco ;
- pixels[12*12] - Array que contém os pixels do bloco;

- s[8][2] - Array que conterá os valores que indicam o grau de dark e bright de cada sub-janela;
- Ridge_max – Variável que conterá o valor máximo entre as diversas orientações possíveis de ridge end para um bloco 12 x 12;
- Ridge_qual – Variável que indicará qual é a orientação do ridge end mais provável de estar no bloco 12 x 12;
- Bif_max – Variável que conterá o valor máximo entre as diversas orientações possíveis de bifurcações para um bloco 12 x 12;
- Bif_qual – Variável que indicará qual é a orientação da bifurcação mais provável de estar no bloco 12 x 12;

Métodos e Funções

- get_pixel (tx, ty) – Retorna o pixel correspondente às coordenadas tx, ty (relativas);
- put_pixel (x, y, value) – Atribui ao pixel de coordenadas relativas x e y o valor value;
- set_pixels(entrada[]) – Captura os pixels e os coloca em entrada[], calcula a média e chama define_fuzzy;
- define_fuzzy() – Calcula a média para cada uma das 8 sub-janelas do bloco 12 x 12, e chama as funções fuzzy_bright e fuzzy_dark para cada delas, guardando em s[][]. Chama também extrai_ridge_max e extrai_bif_max e dependendo dos valores retornados determina que no bloco existe ou não uma minúcia e de que tipo.
- Extrai_ridge_max() – Calcula qual é a orientação de ridge end mais provável para aquele bloco 12 x 12, e coloca o valor em ridge_max e a orientação em ridge_qual;
- Extrai_bif_max() – Calcula qual é a orientação de bifurcação mais provável para aquele bloco 12 x 12, e coloca o valor em bif_max e a orientação em bif_qual;

- Fuzzy_dark (valor) – Aplica uma função fuzzy comparando a média de uma sub-janela com a média do bloco todo. O resultado da função será 1 caso a média local seja bem mais escura que a do bloco, e 0 se o contrário ocorrer. Para valores próximos, há uma interpolação linear entre 0 e 1, cuja inclinação depende de k_fuzzy;
- Fuzzy_bright (valor) – Complementar de Fuzzy_dark. Aplica uma função fuzzy comparando a média de uma sub-janela com a média do bloco todo. O resultado da função será 1 caso a média local seja bem mais clara que a do bloco, e 0 se o contrário ocorrer. Para valores próximos, há uma interpolação linear entre 0 e 1, cuja inclinação depende de k_fuzzy. A soma de Fuzzy_bright e Fuzzy_dark para uma sub-janela sempre é 1.

Abaixo, temos e interface client-side do sistema, com o algoritmo de extração de minúcias operando:

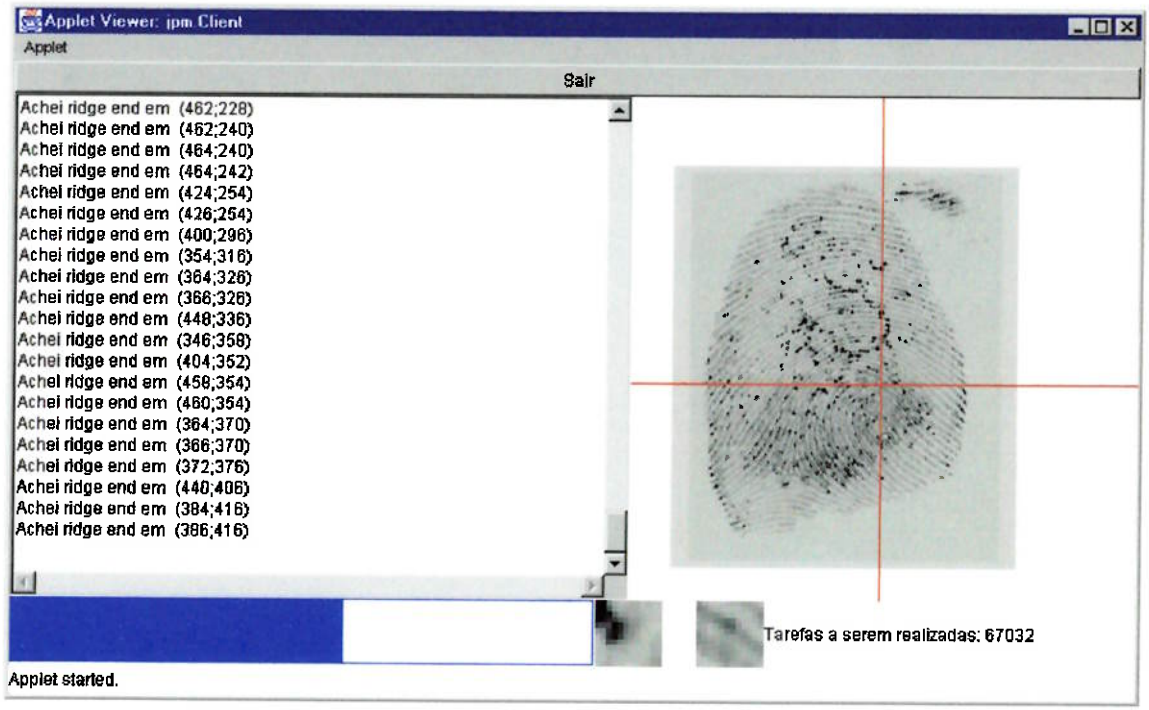


Figura 14 – Interface Client reconhecendo minúcias

6.3 Comparação

A comparação das digitais é feita utilizando um sistema que especifica uma tolerância sobre a posição das minúcias originais e gera regiões onde as minúcias das digitais a serem analizadas deverão estar para que a digital a ser analisada seja correpondente à original.

Na interface do servidor, temos um exemplo deste procedimento, onde os quadrados azuis especificam as regiões das minúcias prévias geradas pela digital original, como mostra a figura:

Figura 15 - Digital original com respectivas minúcias

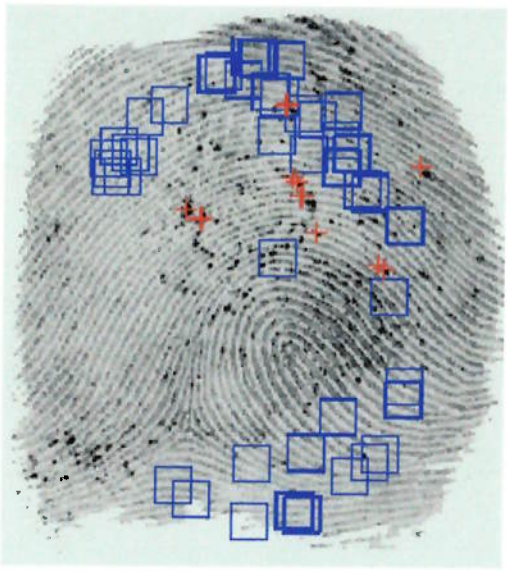


Figura 17 - Digital em que as minúcias não coincidem

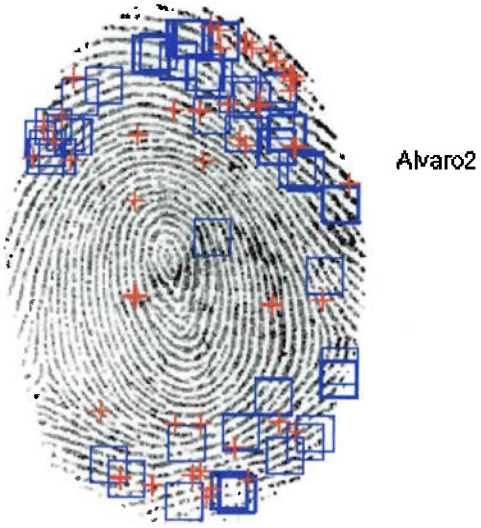


Figura 18 – Digital em que as minúcias estão coincidindo

7. Resultados

Neste capítulo será apresentada uma série de experiências e testes realizada com a versão final do software implementado. Muitos testes, realizados durante todo o desenvolvimento do projeto foram essenciais para se chegar a conclusões relevantes relativas ao desempenho do sistema sob as mais diversas condições e para observar se na prática as ponderações feitas quanto aos algoritmos de distribuição e de identificação de minúcias estavam corretas. Além disso, os testes naturalmente tiveram efeito direto sobre a forma final do software, uma vez que muitas vezes se deparou com imprevistos que ocasionaram revisão de decisões anteriores.

Os seguintes aspectos do desempenho do sistema serão analisados separadamente:

- Sensibilidade do algoritmo de detecção de minúcias e de comparação de impressões digitais à constante k_{limite};
- Efeito do número de processadores no desempenho do sistema (tempos de distribuição de imagens e de processamento puro);
- Desempenho do sistema em Intranets e Internet;
- Performance do sistema para a identificação de minúcias e comparação de impressões digitais.

Em primeiro lugar, deparou-se com o seguinte problema: qual o valor ideal para k_{limite}? Conforme explicado anteriormente, k_{limite} é um valor entre 0 e 1 que é usado para se decidir se um determinado bloco da imagem pode ser considerado como uma minúcia ou não. Desta forma, apura-se o valor máximo entre ridge_max e bif_max e se este valor for superior que k_{limite}, considera-se que naquele bloco existe uma minúcia (cuja natureza é ridge end ou bifurcação dependendo de qual das duas variáveis apresentou o maior valor).

Obviamente este valor é pois um ponto crítico do sistema, uma vez que ele determinará decisivamente o sucesso (ou insucesso) no reconhecimento efetivo de minúcias.

Desta forma, tão menor fosse o valor de k_{limite}, maior seria a "complacência" do sistema quanto ao reconhecimento de minúcias, ou seja, menos restritivo seria o sistema. O ser menos restritivo por um lado poderia significar ser mais sensível às minúcias, e por outro poderia significar ser pouco rígido, reconhecendo como minúcias blocos que não deveriam sê-lo.

Uma maneira encontrada para se tentar otimizar o valor de k_{limite} foi a seguinte: dada uma imagem de impressão digital e um conjunto de nove outras imagens de impressões digitais (sendo que apenas uma entre as nove correspondia à mesma impressão digital considerada separadamente), e usando o critério selecionado para compará-las, o sistema foi testado com diferentes valores de k_{limite} e classificou-se as nove impressões digitais candidatas em ordem crescente de minúcias correspondidas. Naturalmente, a primeira colocada deveria sempre ser àquela que realmente correspondia à impressão digital teste. O critério para se apurar o melhor valor de k_{limite} foi simplesmente observar a distância relativa entre as duas imagens melhor colocadas na classificação. Assim, o melhor k_{limite} forneceria uma restritividade ideal que colocaria a segunda imagem mais parecida com a original o mais longe possível da primeira colocada.

A tabela seguinte mostra os dados coletados nesta experiência:

k_{limite}	Distância relativa entre a 1ª e 2ª impressão digital
0.5	5%
0.6	18%
0.7	50%
0.8	30%
0.9	10%

Para o k_{limite} em 0.5, o sistema reconhecia como minúcias a maior parte do ruído das digitais, gerando um número excessivamente elevado de minucias. Dessa forma, embora a diferença absoluta do número de minúcias reconhecidas para entre a primeira e a segunda digital fosse grande, a diferença relativa era muito pequena (e de fato, a qualidade do sistema era baixa já que quase todas as minúcias eram resultado de ruído).

Aumentando-se o k_{limite}, a qualidade do sistema teve uma tendência a melhorar, já que a razão sinal/ruído decrescia conforme crescia o rigor do algoritmo em considerar uma janela como minúcia. Entretanto, essa qualidade passa por um máximo, pois à partir de um dado k_{limite} (entre 0.7 e 0.8), o sistema ficou tão rigoroso que passou a não considerar minúcias mesmo as minúcias verdadeiras da digital.

Essa análise é melhor observada no gráfico seguinte:

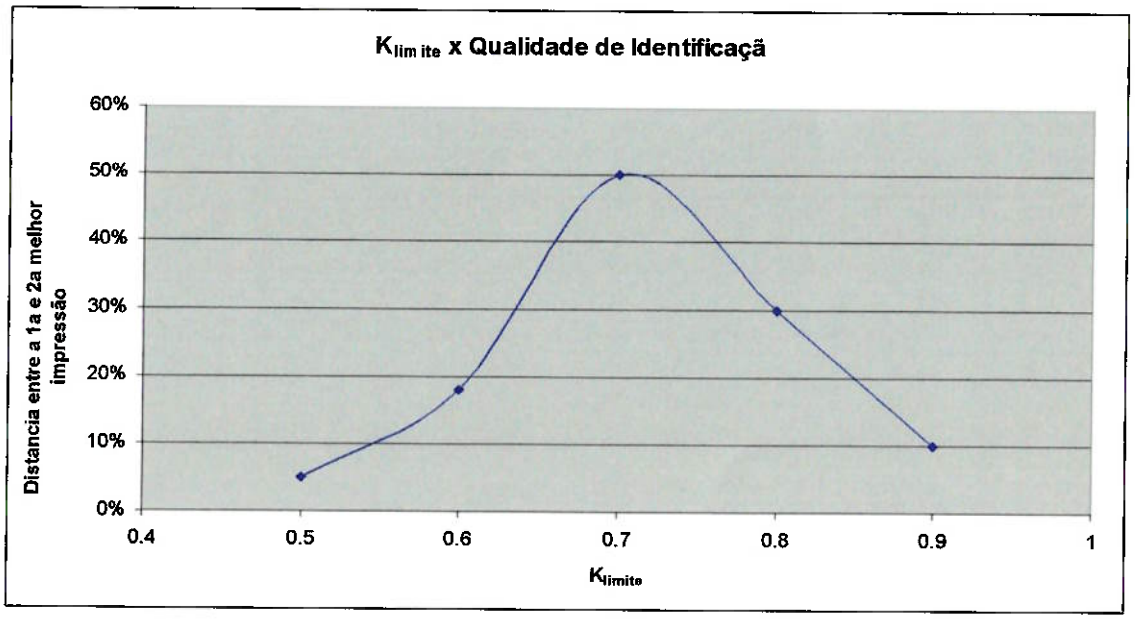


Gráfico 1 – Kfuzzy x Qualidade

Num segundo momento, procedeu-se a seguinte experiência: realizou-se toda a fase de identificação de minúcias em um número variável de processadores (1 a 6) e cronometrou-se os tempos envolvidos (tempo de processamento puro e tempo de distribuição de tarefas). Este teste foi

realizado em uma Intranet composta por 6 Pentium II 400 Mhz. Obteve-se os seguintes resultados:

Núm. De Processadores	Tempo de Distribuição de Tarefas (s)	Tempo de Processamento (s)	Tempo Teórico de Processamento (s)	Tempo Total (s)
1	10	140	140	150
2	11	76	70	87
3	12	56	47	68
4	15	42	35	57
5	17	35	28	52
6	19	31	23	50

Os tempos de distribuição de tarefas entre os diversos processadores foram medidos separadamente dos tempos de processamento real. O tempo de processamento teórico foi calculado baseando-se no tempo de processamento de apenas um processador (140 segundos). Desta forma, o tempo teórico é expresso por (n é o número de processadores):

$$T_{teorico} = \frac{140}{n}$$

O seguinte gráfico mostra os resultados obtidos:

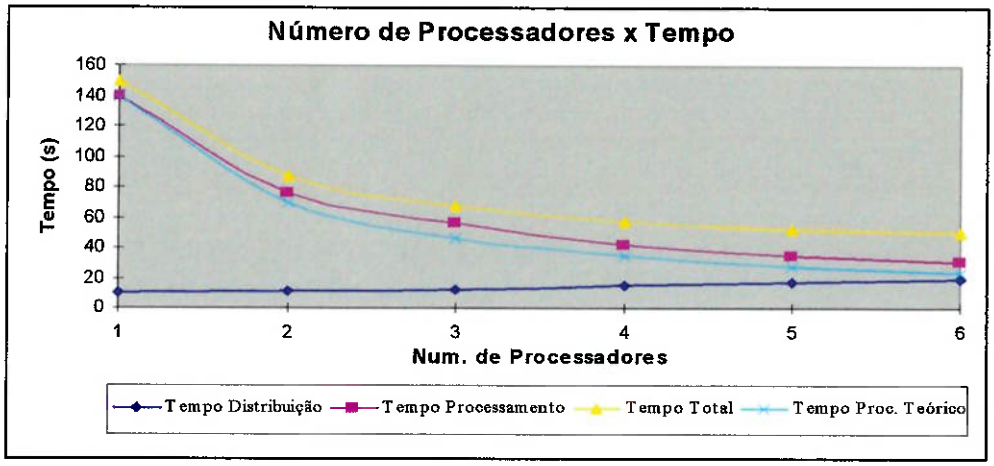


Gráfico 2 - Desempenho do sistema x Número de Processadores

Para esta experiência realizada, estimou-se as velocidades de processamento da seguinte forma:

$$V_{processamento} = \frac{Num.Tarefas}{Tempo}$$

As velocidades de processamento estimadas foram (para um número de aproximadamente 300.000 tarefas realizadas na impressão digital utilizada):

Núm. De Processadores	Tempo Total (s)	Velocidade (tarefas/s)	Velocidade Teórica (tarefas/s)
1	150	2.143	2.143
2	87	3.947	4.286
3	68	5.357	6.429
4	57	7.143	8.571
5	52	8.571	10.714
6	50	9.677	12.857

A curva de "Speed Up" do sistema é mostrada no gráfico seguinte:

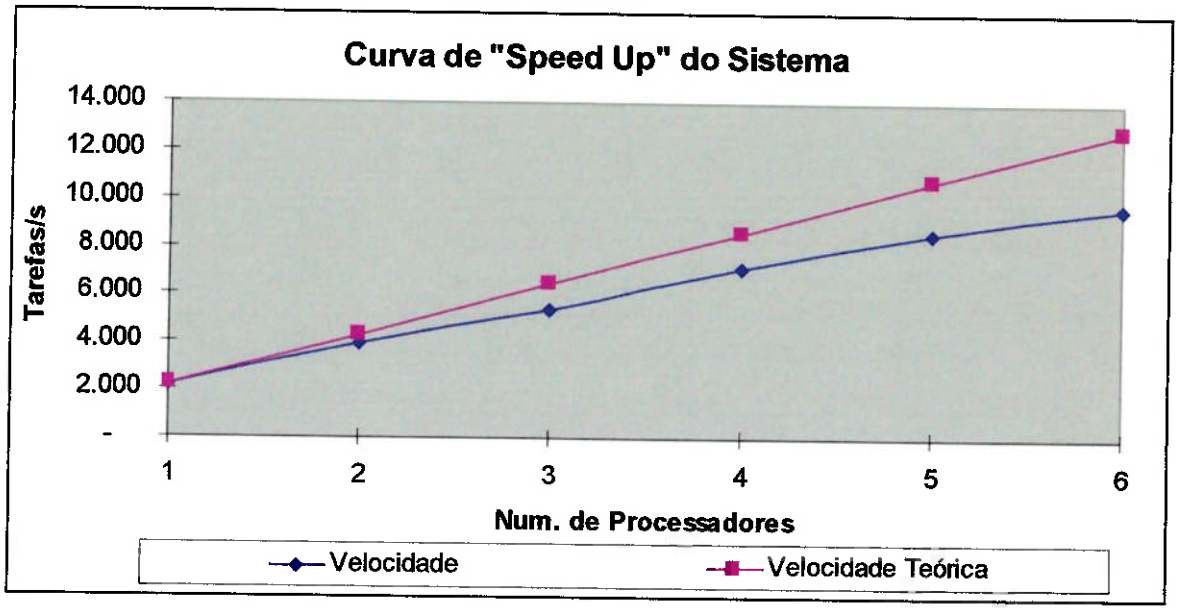


Gráfico 3 - Curva de "Speed Up" do sistema

Uma outra experiência que possibilitou importantes conclusões foi testar o software implementado com quatro máquinas em uma Intranet e uma quinta máquina remota via Internet. Para tanto, foram colocadas 4 máquinas na rede local (Vila Mariana), e uma máquina remota (Paulista). A velocidade de comunicação entre as máquinas da rede local e a máquina remota estava restrita pelo link de 64Kbits/sec que as conectava. Os resultados obtidos foram os seguintes:

Núm. De Processadores	Tempo de Distribuição de Tarefas (s)	Tempo de Processamento (s)	Tempo Total (s)
5 processadores em Intranet	17	35	52
4 em Intranet e 1 via Internet	27	157	184

Observa-se que não só o tempo de distribuição de tarefas foi consideravelmente afetado. O tempo de processamento também foi extremamente influenciado pela diferença de velocidades de comunicação nos dois ambientes diferentes (a velocidade de comunicação na Intranet era de

aproximadamente 3Mb/s enquanto que o link com a Internet era de 64kb/s, ou seja, 48 vezes maior).

Pode-se estimar assim a parcela do tempo considerado de processamento utilizado em comunicação (e consequentemente a parcela utilizada em processamento puro).

Seja t_1 o tempo de processamento das 5 máquinas na Intranet e t_2 o tempo de processamento na condição em que uma das máquinas era remota, utilizando-se da Internet. Assim, t_2-t_1 é a diferença entre os tempos de processamento (total, englobando comunicação e processamento puro) nas duas condições de rede.

Como o tempo de processamento puro é o mesmo nos dois casos (não importa se a máquina é remota, e sim que o processador também era um Pentium II 400 Mhz), conclui-se que t_2-t_1 é a diferença entre os tempos de comunicação de processamento.

Porém, como é conhecida a velocidade de comunicação nas duas condições de rede, tem-se que:

$$\frac{T_2}{T_1} = \frac{3Mb/s}{64Kb/s} = 48$$

Como sabemos que:

$$T_2 - T_1 = 157 - 35 = 122s$$

Pode-se resolver as duas equações anteriores e encontrar os valores dos tempos de comunicação no processamento para os dois casos:

$$T_1 = 2.59s$$

$$T_2 = 124.6s$$

Como temos os tempos de processamento nos dois casos, pode-se calcular o tempo de processamento puro (que não envolve a comunicação) que é:

$$T_P = 157 - 122.3 = 36 - 1.28 = 32.4s$$

No primeiro caso, a relação entre o tempo de comunicação no processamento sobre o tempo total de processamento é:

$$\frac{T_1}{T_{P1}} = \frac{2.59s}{35s} = 7.4\%$$

Em contrapartida, no caso em que a comunicação foi feita através de um link de 64 Kb/s via Internet, esta relação foi de:

$$\frac{T_2}{T_{p2}} = \frac{122.3s}{157s} = 77.9\%$$

Na Intranet, apenas 7.4% do tempo total de processamento foi decorrente de tempo gasto em comunicação enquanto que no caso da Internet, 63.8% do tempo de processamento foi gasto em comunicação.

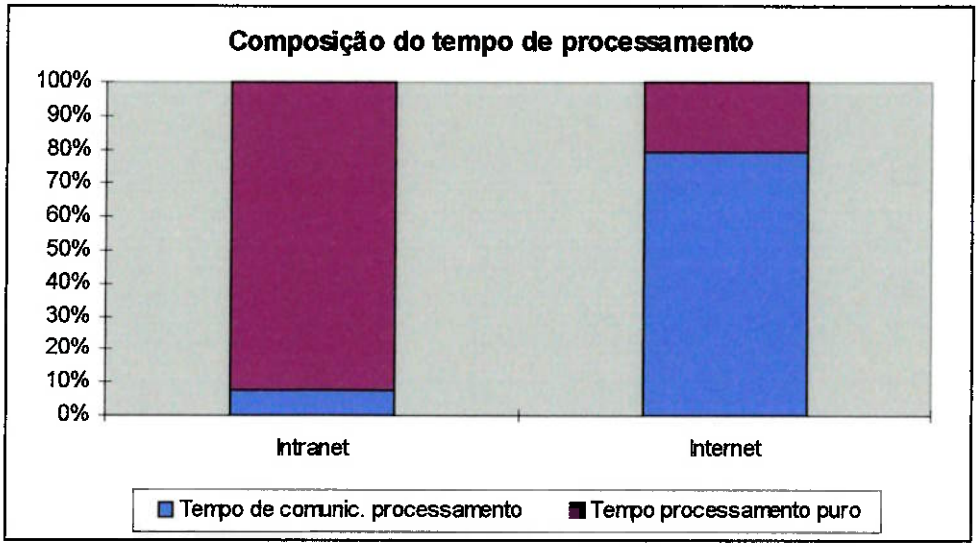


Gráfico 4 - Composição do tempo de processamento

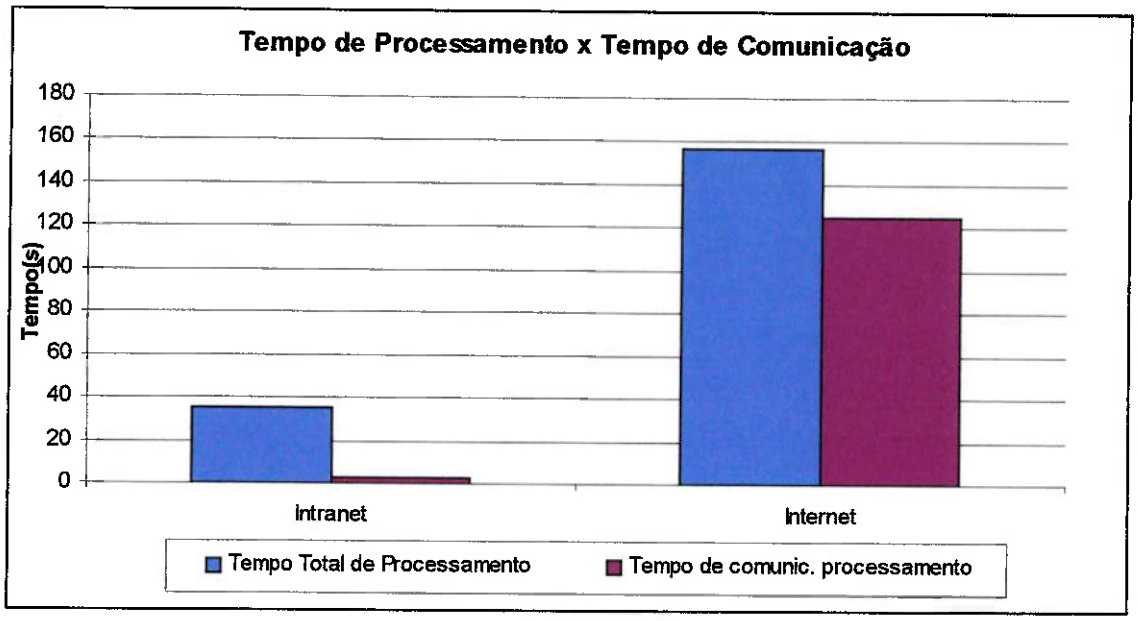
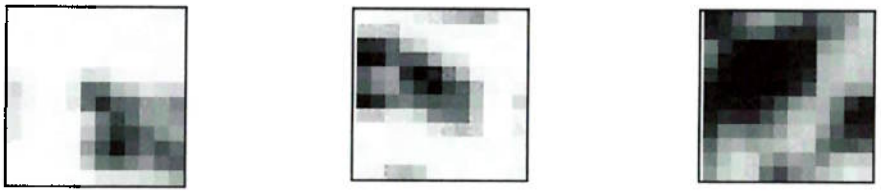


Gráfico 5 - Tempo de Processamento x Tempo gasto em comunicação

Além disso, tivemos que avaliar a performance do sistema no aspecto de capacidade de identificação de minúcias. Seguem algumas imagens de minúcias identificadas como ridge end e bifurcações:



2 "ridge-ends" verdadeiros e um falso (na ordem)

2 bifurcações verdadeiras e uma falsa (na ordem)

Figura 19 – Minúcias reconhecidas

Não há outra maneira para avaliar a capacidade de identificação de minúcias que não a visual. Através da evolução das formas da janela de identificação de minúcias, chegamos a um algoritmo bem sucedido, que nos permitiu, conforme será observado em logo abaixo, um grau confiável de reconhecimento.

Finalmente, o melhor teste para o sistema é verificar se ele realmente é capaz de identificar duas imagens de impressões digitais como sendo de uma mesma pessoa. Para uma base de dados com 25 impressões digitais diferentes, e uma única impressão digital repetida, obtivemos os seguintes resultados:

Imagem	Grau de Coincidência
Impressão resposta	**71**
1	35
2	33
3	29
4	27
5	25
6	23
7	21
8	20
9	20
10	19
11	18
12	18
13	17
14	16
15	15
16	12
17	12
18	11
19	11
20	10
21	6
22	5
23	5
24	3
Média	**19,28**
Desvio Padrão	**13,74**
Probablilidade Erro	**0,008%**

Conforme pode-se observar na tabela acima, obteve-se um grau de confiabilidade de apenas 0,008% de falsa aceitação. O seguinte histograma mostra a distribuição do número de coincidências da base de dados de 25 impressões digitais em relação à impressão teste.

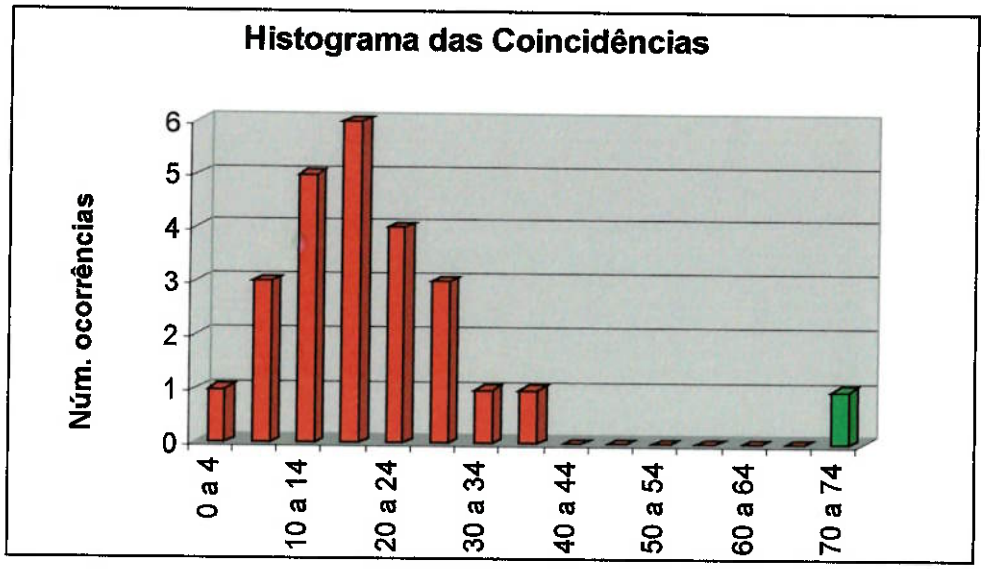


Gráfico 6 – Histograma do número de minúcias coincidentes

8. Conclusões

Os números obtidos relativos à confiabilidade do sistema foram bastante satisfatórios. Considerando a velocidade com que se realiza o reconhecimento de uma única digital (aproximadamente 8s utilizando-se quatro processadores PentiumII 400Mhz), o sistema tem uma boa relação confiabilidade X custo computacional). São conhecidos sistemas com confiabilidade ainda maior (da ordem de 0.001% de probabilidade de falsos reconhecimentos) mas consideravelmente mais custosos (cerca de 30s para a identificação de uma única digital).

O algoritmo mostrou-se viável para aplicações baseadas em redes locais (imensa maioria das redes corporativas), mas comportou-se de forma comprometedora para redes não locais (Extranet's com links de baixa velocidade ou Internet). Para esses ambientes, fica claro que a computação distribuída é, ainda, limitada a aplicações de altíssima granularidade (e portatno, muito restritas).

A curva de speed-up ficou muito próxima da curva teoricamente ideal, e apresentou boa porção aproximável por uma reta, o que a caracteriza como bastante positiva. É bom lembrar que o aspecto desta curva não depende apenas da qualidade do sistema de computação distribuída, mas também da natureza da tarefa a ser realizada. Tarefas pouco granulares (como ray-traces, por exemplo) geram curvas de speed-up muito ruins, e vice-versa.

O algoritmo Fuzzy mostrou-se muito sensível à escolha dos parâmetros internos o que é bom uma vez que, para um ajuste ótimo, o sistema tende a trabalhar muito bem, mas ruim porque dessa forma o sistema tem pouca flexibilidade em se adaptar. Novas formas de captura de digitais, por exemplo, poderão exigir uma etapa de otimização destes parâmetros.

9. Trabalho Futuro

O JPM – Java Parallel Machine, sistema implementado e que foi objeto de estudo do presente trabalho pode servir de base para futuros trabalhos, que podem focar tanto o aspecto da computação distribuída quanto o aspecto do reconhecimento de padrões, mais especificamente na área da biométrica.

No aspecto de computação distribuída, uma série de sugestões para futuras pesquisas, baseadas no JPM, poderiam aqui ser colocadas, tais como:

- Implementação do balanceamento dinâmico de carga, de maneira a enviar uma tarefa (ou um pacote com um número determinado de tarefas) apenas quando o cliente houver completado a(s) tarefa(s) a ele incumbidas;
- Regeneração inteligente, que ocorreria de maneira a tolerar falhas como o desligamento de um processador cliente. O servidor poderia armazenar as tarefas enviadas a cada cliente e em caso de falha, o servidor poderia reenviá-las a outro processador.

Quanto aos algoritmos de reconhecimento de impressões digitais, seria interessante se tentar uma implementação do algoritmo utilizado pelo NIST, com a transformada de Karhunen-Loève na implementação paralela, ou ainda, se tentar uma abordagem inovadora fazendo-se uso de redes neurais. Ainda poder-se-ia sugerir que se propusessem métodos mais sofisticados de otimização de parâmetros como K_{fuzzy} e K_{limite}, que afetam sensivelmente a performance do sistema.

10. Referências Bibliográficas

[1] P. Gladychev, ª Patel, D. O'Mahony, *Cracking RC5 with Java applets*

[2] G. Candela, P. Grother, C. Watson, R. Wilkinson, C. Wilson, *PCASYS – A Pattern-level Classification Automation System for Fingerprints*, Gaithersburg, 1995

[3] J. Stosz, L. Alyea, *Automated system for fingerprint authentication using pores and ridge structure*, Ft. Maede, 1996.

[4] V. Getov, S. Hummel. S. Mintchev, *High-Performance Parallel Programing in Java: Exploiting Native Libraries*, Yorktown Heights, 1997.

[5] ª Alexandrov, M. Ibel, K. Schauser, K. Scheiman, *Super-Web: Research issues in Java-Based Global Computing*, Workshop on Java for High Performance Scientific and Engeneering Computing Simulation and Modelling, Syracuse University, New York, 1996.

[6] G.Candela, R. Chellapa, *Comparative Performence of Classificatiopn Methods for Fingerprints*, Gaithersburg, 1993.

[7] C. Wilson, G. Candela, C. Watson, *Neural Network Fingerprint Classification*, J, Artificial Neural Networks, 1, No. 2, 1993.

[8] C. Wilson, J. Blue, º Omidvar, *Improving Neural Network Performance for Character and Fingerprint Classification by Altering Network Dynamics,* World Congress on Neural Networks Proceeding, Washington DC, 1995.

Anexo

Código-Fonte da JPM

(compilado com JDK1.1.7A)

InterfaceServer.java:

```
package jpm;

import java.rmi.Remote;
import java.rmi.RemoteException;
import java.lang.*;

public interface InterfaceServer extends java.rmi.Remote {
    String sayHello(InterfaceClient objeto) throws RemoteException;
    void estou_saindo(String qual) throws RemoteException;
    void ja_acabei(String qual) throws RemoteException;
    void achei_minucia(int a, int b) throws RemoteException;
}
```

InterfaceClient.java:

```
package jpm;

import java.rmi.*;


public interface InterfaceClient extends java.rmi.Remote {
      /* Interface para o applet poder ter metodos seus chamados */

      void servidor_me_chamou() throws RemoteException;
      void adiciona_tarefa(int a, int b) throws RemoteException;
      void le_imagem(String n_imagem) throws RemoteException;
}
```

Client.java:

```
package jpm;

import java.applet.Applet;
import java.awt.Graphics;
import java.rmi.Naming;
import java.rmi.*;
import java.rmi.server.*;
import java.rmi.RemoteException;
import java.awt.*;
import java.awt.image.*;
import java.util.*;
import java.net.*;
import java.lang.*;
import tom.*;

public class Client extends Applet implements InterfaceClient,
java.io.Serializable {

    public String meu_nome = "";
    private String message = "blank";
    Button botao_sair = new Button("Sair");
    Image imagem_trabalho;
    int[] entrada = new int[832*768];
    boolean chamou_tarefas = false;
    Client_thread meu_thread;
    Display_client meu_painel = new Display_client();


    // "obj" is the identifier that we'll use to refer
    // to the remote object that implements the "Hello"
    // interface
    InterfaceServer obj = null;
/*    Hashtable tarefas = new Hashtable(5000);*/
    Tarefa[] tarefas = new Tarefa[50000];
    int n_tarefas = 0;
    int[] t_x = new int[560000];
    int[] t_y = new int[560000];



      public synchronized void adiciona_tarefa(int a, int b)
      {
/*          tarefas.put(new Integer(n_tarefas), nova_tarefa);*/
          for (int x1=0; x1<12; x1 = x1+2) {
          for (int y1=0; y1<12; y1 = y1+2) {
                t_x[n_tarefas] = a + x1;
                t_y[n_tarefas] = b + y1;
                n_tarefas++;
          }
          }

          if ( (n_tarefas%19)==0 ) {
                meu_painel.n_tarefas(n_tarefas);
          }
```

```
            if (n_tarefas>559900) {
                  System.out.println("Acabou capacidade "+n_tarefas);
            }
      }

      public void servidor_me_chamou()
      {
            meu_painel.manda_mensagem("Ordem de inicio do servidor");
            chamou_tarefas = true;
            repaint();
      }

      public void inicia_tarefas()
      {
            for (int i=0; i<n_tarefas; i++) {
                  processa_tarefa(t_x[i],t_y[i]);
                  meu_painel.estou_na_tarefa(i,t_x[i]/2,t_y[i]/2);
            }

            n_tarefas = 0;
            /* Manda mensagem dizendo que acabou suas tarefas */
             try {
                  obj.ja_acabei(meu_nome);
             } catch (Exception e) {};
      }

      public boolean action(Event evt, Object what) {
            if (evt.target == botao_sair) {
                  meu_painel.manda_mensagem("Processador extinto -
disposeALL");
                  if (obj != null) {
                   try {
                        obj.estou_saindo(meu_nome);
                   } catch (Exception e) {};
                  }
            }
            return(false);
      }



    public void init() {
      /* Exportar o applet como objeto remoto! */

      this.setLayout(new BorderLayout());

      add("North", botao_sair);
      add("Center", meu_painel);


      meu_painel.manda_mensagem("ClientThread inicializado");
      meu_thread = new Client_thread(this);

      try {
            meu_painel.manda_mensagem("Exportando esse applet como remoto");
            UnicastRemoteObject.exportObject(this);
      } catch (Exception e) {
```

```
            meu_painel.manda_mensagem("Erro ao exportar applet como remoto:
");
            e.printStackTrace();
        }
        /* Exportar o applet como objeto remoto! */
        try {
            obj = (InterfaceServer)Naming.lookup("//" +
                    getCodeBase().getHost() + "/Server");
              meu_painel.manda_mensagem("Servidor localizado");
            meu_nome = obj.sayHello(this);
        } catch (Exception e) {
            System.out.println("HelloApplet exception: " +
                               e.getMessage());
            e.printStackTrace();
        }
    }

    public void le_imagem(String n_imagem)
    {
      MediaTracker tracker;


      n_imagem = "imagens/" + n_imagem;

      System.out.println("Carregando imagem do servidor");
      meu_painel.manda_mensagem("Carregando imagem do servidor");
      System.out.println("Carregando imagem: " + n_imagem);

      tracker = new MediaTracker(this);
      imagem_trabalho = getImage(getCodeBase(), n_imagem);
      tracker.addImage(imagem_trabalho, 0);
      try {
            tracker.waitForID(0);
      } catch (InterruptedException e) {
            System.out.println("Erro no mediatracker do client");
            return;
      }

      meu_painel.n_img_dig(imagem_trabalho);
      meu_painel.manda_mensagem("Imagem carregada");

      try {
            PixelGrabber pg = new
PixelGrabber(imagem_trabalho,0,0,832,768,entrada,0,832);
           pg.grabPixels();
      } catch (Exception e) {}


      meu_thread.start();

      meu_painel.manda_mensagem("Client_Thread inicializado");
      repaint();
    }

    public void paint(Graphics g) {
/*    g.drawString(message, 25, 50); */
/*    if (imagem_trabalho != null) {
```

```
        g.drawImage(imagem_trabalho, 0,10,104,96,this);
      }
*/    }


      synchronized void processa_tarefa(int a, int b) {
       int[] n_ent = new int[12*12];

       Bloco_minucia n_minu = new Bloco_minucia();

       int quantas_minucias = 0;

       for (int j=0; j<12; j++) {
       for (int i=0; i<12; i++) {
         n_ent[i + 12*j] = entrada[(a+i) + 832*(b+j)];
       }
       }
       n_minu.set_pixels(n_ent);

       Image nova_img = getToolkit().createImage (new
MemoryImageSource(12,12,n_ent,0,12));

      meu_painel.seta_janela(nova_img.getScaledInstance(48,48,Image.SCALE_FA
ST));

      if ( (n_minu.ridge_max > 0.92f)) {
            meu_painel.manda_mensagem("Achei ridge end em \t(" + a + ";" + b
+ ")");
            try {
               obj.achei_minucia(a,b);
            } catch (Exception e) {
               System.out.println("Pau em alguma coisa");
            }
            quantas_minucias++;

     meu_painel.seta_janela_fixa(nova_img.getScaledInstance(48,48,Image.SCA
LE_FAST));

            n_minu.novo_frame(quantas_minucias, quantas_minucias, this);

      } else {
            if (n_minu.bif_max > 0.92f) {
              meu_painel.manda_mensagem("Achei bifurcacao em \t(" + a + ";"
+ b + ")");
              quantas_minucias++;
              n_minu.novo_frame(quantas_minucias, quantas_minucias, this);

meu_painel.seta_janela_fixa(nova_img.getScaledInstance(48,48,Image.SCALE_FAS
T));

            } else {
            }
      }
    }
}
```

```
class Bloco_minucia {



        float k_fuzzy=25.0f;
        float k_limite=0.75f;
        Frame meu_frame;


        public boolean sobrevivi=false;
        public int med_value;
        public int x,y;

        public float ridge_max=0;
        public int   ridge_qual=0;
        public float bif_max=0;
        public int   bif_qual=0;


        public int[] pixels;

        float[][] s;


        public void novo_frame(int a, int b, Object componente) {

              int temp=0;
              int s=0;
              for (int j=0; j< 12; j++) {
               for (int i=0; i<12; i++) {
                    temp = (pixels[s])&0x000000ff;
                    pixels[s] = 0xff000000
                          + 256*(256*temp) +
                            256*temp +
                            temp;
                    s++;
               }
              }


              meu_frame = new Frame("Minucia");
              meu_frame.setBounds(a,b,70,130);

              Image nova = meu_frame.getToolkit().createImage(new

              MemoryImageSource(12,12,pixels,0,12));
              Canvas_img n_canvas = new Canvas_img(nova, 48, 48);
              meu_frame.add("Center", n_canvas);
/*            meu_frame.show();*/


        }


        Bloco_minucia() {
          med_value = 0;
```

```
    x=0;
    y=0;
    pixels = new int[12*12];
    s = new float[8][2];
  }

  int get_pixel(int tx, int ty) {
        return(pixels[ty*12 + tx]);
  }

  void put_pixel(int x, int y, int value) {
        pixels[y*12 + x] = value;
  }


  public void set_pixels(int[] entrada) {
        ColorModel cm = ColorModel.getRGBdefault();

        med_value = 0;

        int s=0;
        for (int j=0; j< 12; j++) {
         for (int i=0; i<12; i++) {
              pixels[s] = cm.getRed(entrada[s]);
              med_value += pixels[s];
              s++;
         }
        }
        med_value /= 144;

        define_fuzzy();

  }

  void define_fuzzy() {
        int media;

        media =
              (get_pixel(2,0) +
              get_pixel(3,0) +
              get_pixel(4,0) +
              get_pixel(5,0) +
              get_pixel(6,0) +
              get_pixel(7,0) +
              get_pixel(8,0) +
              get_pixel(9,0) +
              get_pixel(3,1) +
              get_pixel(4,1) +
              get_pixel(5,1) +
              get_pixel(6,1) +
              get_pixel(7,1) +
              get_pixel(8,1) +
              get_pixel(3,2) +
              get_pixel(4,2) +
              get_pixel(5,2) +
              get_pixel(6,2) +
              get_pixel(7,2) +
```

```
        get_pixel(8,2) +
        get_pixel(4,3) +
        get_pixel(5,3) +
        get_pixel(6,3) +
        get_pixel(7,3) +
        get_pixel(5,4) +
        get_pixel(6,4)  )/  26;

 s[0][0] = fuzzy_dark(media);
 s[0][1] = fuzzy_bright(media);


 media =
        (get_pixel(11,2) +
        get_pixel(11,3) +
        get_pixel(11,4) +
        get_pixel(11,5) +
        get_pixel(11,6) +
        get_pixel(11,7) +
        get_pixel(11,8) +
        get_pixel(11,9) +
        get_pixel(10,3) +
        get_pixel(10,4) +
        get_pixel(10,5) +
        get_pixel(10,6) +
        get_pixel(10,7) +
        get_pixel(10,8) +
        get_pixel(9,3) +
        get_pixel(9,4) +
        get_pixel(9,5) +
        get_pixel(9,6) +
        get_pixel(9,7) +
        get_pixel(9,8) +
        get_pixel(8,4) +
        get_pixel(8,5) +
        get_pixel(8,6) +
        get_pixel(8,7) +
        get_pixel(7,5) +
        get_pixel(7,6)  )/  26;

 s[2][0] = fuzzy_dark(media);
 s[2][1] = fuzzy_bright(media);

 media =
        (get_pixel(2,11) +
        get_pixel(3,11) +
        get_pixel(4,11) +
        get_pixel(5,11) +
        get_pixel(6,11) +
        get_pixel(7,11) +
        get_pixel(8,11) +
        get_pixel(9,11) +
        get_pixel(3,10) +
        get_pixel(4,10) +
        get_pixel(5,10) +
        get_pixel(6,10) +
        get_pixel(7,10) +
```

```
        get_pixel(8,10) +
        get_pixel(3,9) +
        get_pixel(4,9) +
        get_pixel(5,9) +
        get_pixel(6,9) +
        get_pixel(7,9) +
        get_pixel(8,9) +
        get_pixel(4,8) +
        get_pixel(5,8) +
        get_pixel(6,8) +
        get_pixel(7,8) +
        get_pixel(5,7) +
        get_pixel(6,7)  )/  26;

 s[4][0] = fuzzy_dark(media);
 s[4][1] = fuzzy_bright(media);

 media =
        (get_pixel(0,2) +
        get_pixel(0,3) +
        get_pixel(0,4) +
        get_pixel(0,5) +
        get_pixel(0,6) +
        get_pixel(0,7) +
        get_pixel(0,8) +
        get_pixel(0,9) +
        get_pixel(1,3) +
        get_pixel(1,4) +
        get_pixel(1,5) +
        get_pixel(1,6) +
        get_pixel(1,7) +
        get_pixel(1,8) +
        get_pixel(2,3) +
        get_pixel(2,4) +
        get_pixel(2,5) +
        get_pixel(2,6) +
        get_pixel(2,7) +
        get_pixel(2,8) +
        get_pixel(3,4) +
        get_pixel(3,5) +
        get_pixel(3,6) +
        get_pixel(3,7) +
        get_pixel(4,5) +
        get_pixel(4,6)  )/  26;

s[6][0] = fuzzy_dark(media);
s[6][1] = fuzzy_bright(media);

media =
        (get_pixel(0,0) +
        get_pixel(0,1) +
        get_pixel(0,2) +
        get_pixel(0,3) +
        get_pixel(1,0) +
        get_pixel(1,1) +
        get_pixel(1,2) +
        get_pixel(1,3) +
```

```
        get_pixel(1,4) +
        get_pixel(2,0) +
        get_pixel(2,1) +
        get_pixel(2,2) +
        get_pixel(2,3) +
        get_pixel(2,4) +
        get_pixel(3,0) +
        get_pixel(3,1) +
        get_pixel(3,2) +
        get_pixel(3,3) +
        get_pixel(3,4) +
        get_pixel(4,1) +
        get_pixel(4,2) +
        get_pixel(4,3) +
        get_pixel(4,4)  )/  23;

 s[7][0] = fuzzy_dark(media);
 s[7][1] = fuzzy_bright(media);


 media =
       (get_pixel(7,0) +
       get_pixel(7,1) +
       get_pixel(7,2) +
       get_pixel(7,3) +
       get_pixel(8,0) +
       get_pixel(8,1) +
       get_pixel(8,2) +
       get_pixel(8,3) +
       get_pixel(8,4) +
       get_pixel(9,0) +
       get_pixel(9,1) +
       get_pixel(9,2) +
       get_pixel(9,3) +
       get_pixel(9,4) +
       get_pixel(10,0) +
       get_pixel(10,1) +
       get_pixel(10,2) +
       get_pixel(10,3) +
       get_pixel(10,4) +
       get_pixel(11,1) +
       get_pixel(11,2) +
       get_pixel(11,3) +
       get_pixel(11,4)  )/  23;

s[5][0] = fuzzy_dark(media);
s[5][1] = fuzzy_bright(media);


media =
      (get_pixel(7,7) +
      get_pixel(7,8) +
      get_pixel(7,9) +
      get_pixel(7,10) +
      get_pixel(8,7) +
      get_pixel(8,8) +
      get_pixel(8,9) +
```

```
                get_pixel(8,10) +
                get_pixel(8,11) +
                get_pixel(9,7) +
                get_pixel(9,8) +
                get_pixel(9,9) +
                get_pixel(9,10) +
                get_pixel(9,11) +
                get_pixel(10,7) +
                get_pixel(10,8) +
                get_pixel(10,9) +
                get_pixel(10,10) +
                get_pixel(10,11) +
                get_pixel(11,8) +
                get_pixel(11,9) +
                get_pixel(11,10) +
                get_pixel(11,11)  )/  23;

            s[3][0] = fuzzy_dark(media);
            s[3][1] = fuzzy_bright(media);

            media =
                (get_pixel(0,7) +
                get_pixel(0,8) +
                get_pixel(0,9) +
                get_pixel(0,10) +
                get_pixel(1,7) +
                get_pixel(1,8) +
                get_pixel(1,9) +
                get_pixel(1,10) +
                get_pixel(1,11) +
                get_pixel(2,7) +
                get_pixel(2,8) +
                get_pixel(2,9) +
                get_pixel(2,10) +
                get_pixel(2,11) +
                get_pixel(3,7) +
                get_pixel(3,8) +
                get_pixel(3,9) +
                get_pixel(3,10) +
                get_pixel(3,11) +
                get_pixel(4,8) +
                get_pixel(4,9) +
                get_pixel(4,10) +
                get_pixel(4,11)  )/  23;

            s[1][0] = fuzzy_dark(media);
            s[1][1] = fuzzy_bright(media);

            extrai_ridge_max();
            extrai_bif_max();

            if ( (ridge_max > k_limite) ||
                 (bif_max > k_limite) ) {
                sobrevivi = true;
/*              System.out.print("              Sobrevivi - (");
                System.out.print(x);
                System.out.print(",");
```

```
                System.out.print(y);
                System.out.print(")    -     ridge_max: ");
                System.out.print(ridge_max);
                System.out.print("       bif_max: ");
                System.out.println(bif_max);
*/          } else {
/*              sobrevivi = false;
                System.out.print("Nao  Sobrevivi - (");
                System.out.print(x);
                System.out.print(",");
                System.out.print(y);
                System.out.print(")    -     ridge_max: ");
                System.out.print(ridge_max);
                System.out.print("       bif_max: ");
                System.out.println(bif_max);
*/          }
      }

      float min(float x1, float x2, float x3, float x4, float x5, float x6)
{
            float resultado;

            resultado = x1;
            if (x2 < resultado) {
                  resultado = x2;
            }
            if (x3 < resultado) {
                  resultado = x3;
            }
            if (x4 < resultado) {
                  resultado = x4;
            }
            if (x5 < resultado) {
                  resultado = x5;
            }
            if (x6 < resultado) {
                  resultado = x6;
            }
            return(resultado);
      }

      void extrai_ridge_max() {
            float valor;

            for (int n=0; n< 8; n++) {
              valor = min (
                        s[n][0],
                        s[(n+2)%8][1],
                        s[(n+3)%8][1],
                        s[(n+4)%8][1],
                        s[(n+5)%8][1],
                        s[(n+6)%8][1]
                      );
              if (valor >= ridge_max) {
                  ridge_max = valor;
                  ridge_qual = n;
              }
```

```
        }
    }

    void extrai_bif_max() {
        float valor;

        for (int n=0; n< 8; n++) {
          valor = min (
                    s[n][1],
                    s[(n+2)%8][0],
                    s[(n+3)%8][0],
                    s[(n+4)%8][0],
                    s[(n+5)%8][0],
                    s[(n+6)%8][0]
                );
          if (valor >= bif_max) {
              bif_max = valor;
              bif_qual = n;
          }
        }
    }

    float fuzzy_dark(int valor) {
        float retorno;

        float media;
        float value;

        media = med_value;
        value = valor;



        if ( valor < (med_value - k_fuzzy) ) {
              retorno = 1;
        } else {
         if ( valor > (med_value + k_fuzzy) ) {
              retorno = 0;
         } else {
              retorno = 0.5f -
                    (value - media)/(2*k_fuzzy);
         }
        }
        return(retorno);
    }
    float fuzzy_bright(int valor) {
        float retorno;

        if ( valor < (med_value - k_fuzzy) ) {
              retorno = 0;
        } else {
        if ( valor > (med_value + k_fuzzy) ) {
              retorno = 1;
        } else {
              retorno = 0.5f + (valor - med_value)/(2*k_fuzzy);
        }
        }
```

```
            return(retorno);
        }
    }


    class Canvas_img extends Canvas {
        Image imagem;
        int w,h;
        int marca_x=0;int marca_y=0;
        boolean marcas=false;

        Canvas_img(Image n_img, int a, int b) {
            this.imagem = n_img;
            this.w = a;
            this.h = b;
            repaint();
        }

        public void marca(int x, int y) {
            if ( (x==0) && (y==0)) {
                marcas = false;
            } else {
                marcas = true;
            }
            marca_x = x;
            marca_y = y;
            repaint();
        }

        public void update(Graphics g) {
            paint(g);
        }

        public void paint(Graphics g) {
            if (imagem!=null) {
                g.drawImage(imagem,0,0,w,h,this);
            }
            if (marcas == true) {
                g.setColor(Color.red);
                g.drawLine(marca_x,0,marca_x,h);
                g.drawLine(0,marca_y,w,marca_y);
            }
        }

        public void update_img(Image n_img) {
            imagem = n_img;
            repaint();
        }
}

class Meu_gauge extends Canvas {

        int w; int h; int total; int atual;
        int coord_atual=0;
        Graphics offscreenG;
        Image offscreenImg;
```

```
        boolean nunca_passei = true;
        boolean reseta=false;

        Meu_gauge(int a, int b, int n_total) {
              this.w = a;
              this.h = b;
              this.total = n_total;
              resize(w,h);
              System.out.println("Passei por aqui 1");
              this.offscreenImg = createImage(416, 40);
              System.out.println("Passei por aqui 2");

 /*           repaint();*/
        }

        public void set_max(int maximo) {
              total = maximo;
              reseta=true;
              repaint();
        }

        public void update(Graphics g) {
              paint(g);
        }

        public void reseta() {
              reseta=true;
              repaint();
        }
        public void paint(Graphics g) {
              if (reseta==true) {
                    reseta=false;
                    g.setColor(Color.white);
                    g.fillRect(0,0,w,h);
                    g.setColor(Color.blue);
              }
               g.setColor(Color.blue);
               g.drawRect(0,0,w-2,h-2);
               g.fillRect(0,0,coord_atual,h-2);
        }

        public void andei(int onde_estou) {
              coord_atual = (onde_estou * w)/total;
              repaint();
        }


}


class Client_thread extends Thread {
        Client meu_dono;

        Client_thread(Client n_meu_dono) {
              this.meu_dono = n_meu_dono;
        }
```

```
        public void run() {
              while(true) {
                    if (meu_dono.chamou_tarefas == true) {
                          meu_dono.chamou_tarefas = false;
                          meu_dono.inicia_tarefas();
                    }
                    try {
                          Thread.sleep(500);
                    } catch (InterruptedException e) {
                          e.printStackTrace();
                    }
              }

        }
}

class Display_client extends Panel {

        TextArea    textos;
        Canvas_img  imagem_digital;
        Label       l_tarefas;
        Meu_gauge   meu_gauge;
        Canvas_img  janela;
        Canvas_img  janela_fixa;

        Display_client() {
              textos = new TextArea();

              imagem_digital = new Canvas_img(null, 416, 384);
              imagem_digital.resize(416,384);
              janela = new Canvas_img(null, 48,48);
              janela.resize(48,48);

              janela_fixa = new Canvas_img(null, 48,48);
              janela_fixa.resize(48,48);

              l_tarefas = new Label("Todos os processadores aguardando
tarefas");
              l_tarefas.resize(230,48);
              meu_gauge = new Meu_gauge(416,48, 0);

              this.setLayout(new BorderLayout());

              this.add("West", textos);
              this.add("East", imagem_digital);

              Panel p1 = new Panel(new BorderLayout());
              p1.add("East", l_tarefas);
              Panel p3 = new Panel(new BorderLayout());
              p3.add("East", janela);
              p3.add("West", janela_fixa);
              p1.add("Center", p3);
              p1.add("West", meu_gauge);
              this.add("South", p1);
        }

        public void manda_mensagem(String mensagem) {
```

```
            textos.append(mensagem + "\n");
        }

        public void seta_janela(Image n_imagem) {
            janela.update_img(n_imagem);
        }

        public void seta_janela_fixa(Image n_imagem) {
            janela_fixa.update_img(n_imagem);
        }

        public void n_tarefas(int n) {
            l_tarefas.setText("Tarefas a serem realizadas: " + n);
            meu_gauge.set_max(n);
        }

        public void estou_na_tarefa(int n, int m_x, int m_y) {
            if (n < 100) {
                meu_gauge.reseta();
            }
            meu_gauge.andei(n);
            imagem_digital.marca(m_x, m_y);
        }

        public void n_img_dig(Image nova) {
            imagem_digital.update_img(nova);
        }
}
```

Server.java:

```
package jpm;

import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;
import java.util.*;
import java.awt.*;

public class Server extends UnicastRemoteObject
    implements InterfaceServer {

      int   tole=18;
      int   passagem;
      public Display_grafico meu_frame;
      String imagem_atual;
      int[] coincidencias = new int[6];
      int[] m_x = new int[2000];
      int[] m_y = new int[2000];
      int minucias_0=0;
      int centroide_x = 0;
      int centroide_y = 0;
      int quantos_acabaram=0;

      Tarefa tarefa_atual;

      Hashtable objetos_remotos = new Hashtable(100);
      lista_tarefa lista_de_tarefa = new lista_tarefa();

      int n_tarefas=0;

      int k=0;

   public Server() throws RemoteException {
       super();
       passagem = 0;
   }

   public void centroide_em(int a, int b) {
     centroide_x = a;
     centroide_y = b;
   }

   public String sayHello(InterfaceClient objeto) throws RemoteException {

     String atual;

     for (int t=0; t<6; t++) {
           coincidencias[t] = 0;
     }
     String retorno = new String("Hello World! - ");
     retorno = retorno.concat(String.valueOf(passagem));
     System.out.print("Inseri o de numero: ");
     atual = String.valueOf(objetos_remotos.size());
```

```
        System.out.println(atual);
        objetos_remotos.put(atual, objeto);
        passagem++;
        meu_frame.set_n_conectados(objetos_remotos.size(), "Mais uma
  conexao");
        return atual;
      }

      public void marca_areas() {
        meu_frame.g_imagem.setColor(Color.blue);
        for (int t=1; t<=minucias_0; t++ ) {
          meu_frame.g_imagem.drawRect( (m_x[t] + centroide_x - tole)/2,
                                       (m_y[t] + centroide_y - tole)/2,
                                       (tole),
                                       (tole)
                                      );
        }
        meu_frame.g_imagem.setColor(Color.black);
      }


      public synchronized void achei_minucia(int a, int b)
      {
        meu_frame.minucia_em(a,b);
        a -= centroide_x;
        b -= centroide_y;

        if (k==0) {
              minucias_0++;
              m_x[minucias_0] = a;
              m_y[minucias_0] = b;
        } else {
              for (int t=1; t<=minucias_0; t++) {
                    if ( (a >= (m_x[t] - tole) ) &&
                         (a <= (m_x[t] + tole) ) &&
                         (b >= (m_y[t] - tole) ) &&
                         (b <= (m_y[t] + tole) )
                       )
                    {
                          coincidencias[k]++;
                    }

              }
        }
      }


      public void estou_saindo(String qual) throws RemoteException {
        meu_frame.manda_mensagem("O processador " + qual + " desconectou.");

      }

      public void ja_acabei (String qual) throws RemoteException {
        quantos_acabaram++;
        meu_frame.manda_mensagem("O processador " + qual + " acabou suas
  tarefas.");
        if (quantos_acabaram==objetos_remotos.size()) {
```

```
        quantos_acabaram=0;
        meu_frame.final_tarefas.setState(true);

        k++;
        if (k<=5) {
         n_tarefas = 0;
         meu_frame.reseta_checks();
         meu_frame.distribui_img.setState(true);

         meu_frame.le_imagem("bd_dig0" + String.valueOf(k) + ".gif");
         meu_frame.manda_mensagem("Analisando impressao: bg_dig0" +
String.valueOf(k) + ".gif");
         distribui_img();
         meu_frame.segmenta();
         meu_frame.atualiza_imagem();
         marca_areas();


         for (int i=0; i<objetos_remotos.size(); i++) {
             try {
                   if (objetos_remotos.containsKey(String.valueOf(i))) {

((InterfaceClient)objetos_remotos.get(String.valueOf(i))).servidor_me_chamou
();
                   } else {
                         System.out.println("Nao contem o objeto");
                   }
             } catch (java.rmi.RemoteException e) {};
         }
        } else {
             for (int t=1; t<=5; t++) {
                   meu_frame.manda_mensagem(
                      "Resultado para bloco0" + t + ".gif -> " +
                      coincidencias[t]);
             }
         }
        }

    }


    public static void main(String args[]) {

        // Create and install a security manager
        if (System.getSecurityManager() == null) {
          System.setSecurityManager(new RMISecurityManager());
        }
        try {
          Server obj = new Server();
          // Bind this object instance to the name "HelloServer"
          Naming.rebind("Server", obj);
          System.out.println("Servidor ligado ao registry");
          obj.meu_frame = new Display_grafico();
          obj.meu_frame.meu_dono = obj;

        } catch (Exception e) {
          System.out.println("Server err: " + e.getMessage());
```

```
            e.printStackTrace();
        }
    }


    public void starta_processos() {
      meu_frame.reseta_checks();
      meu_frame.distribui_img.setState(true);

      meu_frame.le_imagem(imagem_atual);
      distribui_img();
      meu_frame.segmenta();
      meu_frame.atualiza_imagem();


      meu_frame.em_paralelo.setState(true);
      for (int i=0; i<objetos_remotos.size(); i++) {
            try {
                  System.out.println("Entrei no loop de objetos remotos");
                  if (objetos_remotos.containsKey(String.valueOf(i))) {
                   System.out.println("Contem");

((InterfaceClient)objetos_remotos.get(String.valueOf(i))).servidor_me_chamou
();
                   System.out.println("Chamei processo remoto");
                  } else {
                        System.out.println("Nao contem o objeto");
                  }
            } catch (java.rmi.RemoteException e) {};
      }
    }


    public void distribui_img()
    {
      for (int i=0; i<objetos_remotos.size(); i++) {
            try {
                  System.out.println("Entrei no loop de distribuir imgs");
                  if (objetos_remotos.containsKey(String.valueOf(i))) {

((InterfaceClient)objetos_remotos.get(String.valueOf(i))).le_imagem(meu_fram
e.file_in);
                   System.out.println("Imagem distribuida");
                  } else {
                        System.out.println("Processador desconectado!");
                  }
            } catch (java.rmi.RemoteException e) {};
      }

    }


    public synchronized void adiciona_tarefa(int a, int b) {

      int qual_processador = n_tarefas % objetos_remotos.size();
      try {
```

```
            if
(objetos_remotos.containsKey(String.valueOf(qual_processador))) {

      ((InterfaceClient)objetos_remotos.get(String.valueOf(qual_processador)
)).adiciona_tarefa(a,b);
            } else {
                  System.out.println("Processador desconectados!");
            }
      } catch (java.rmi.RemoteException e) {};
      n_tarefas++;
    }
}
```