Sistemas Distribuídos: conceitos fundamentais e técnicas para implementações em Java RMI

Pastor, Luis Paulo R.
Bacharelado em Ciências da Computação / UNIVEM
luispaulo_pastor@hotmail.com
Duarte, Mauricio
Prof. FATEC – Garça / UNIVEM
maur.duarte@gmail.com

Resumo **-** A computação paralela difundida com sistemas distribuídos pode trazer grandes benefícios, como o ganho de desempenho que a computação paralela proporciona, e a possibilidade da utilização de recursos espalhados pelo sistema, que é uma característica dos sistemas distribuídos, além de outras vantagens que essas duas áreas oferecem. Para a comunicação entre os computadores do sistema é utilizado um mecanismo que oferece aos programadores ferramentas necessárias para a programação distribuída, o Java RMI. A comunicação entre os computadores possibilita a programação paralela e dentre os diversos algoritmos em que esta pode ser aplicada, os algoritmos de criptografia foram selecionados para se verificar o ganho de desempenho nos seus processamentos.

Keywords – *Computação Paralela, Java RMI, Algoritmos de Criptografia, Sistemas Distribuídos.*

Abstract **-** The parallel computing diffused with distributed systems can bring great benefits, such as the performance gains that the parallel computing provides, and the possibility of the use of resources scattered throughout the system, that is a feature of distributed systems, besides others advantages that these two areas offer. For communication between computers is used a mechanism that gives developers tools needed for programming distributed, the Java RMI. Communication between computers enables parallel programming and among several parallel programming algorithms can be applied, encryption algorithms were selected to verify the performance gain in their processing.

Keywords *– Parallel Computing, Java RMI, Encryption Algorithms, Distributed Systems.*

I. INTRODUÇÃO

Grandes desafios computacionais em vários segmentos da computação desafiam os profissionais desta área que é cada vez mais crescente pelo mundo todo. Sempre houve uma maneira de resolver problemas que estavam na frente de pesquisadores naquele exato momento ao longo da história, ou por necessidade daquele contexto histórico, ou pelo simples prazer de criar algo novo que fosse mudar o dia-a-dia de pessoas "normais". Com o passar do tempo e a evolução dos computadores, vem aumentando o grau dos desafios que a computação tenta solucionar, e computadores que antes eram suficientes se tornaram incapazes de ajudar. Uma solução financeiramente viável e

interessante é o uso de sistemas distribuídos com processamento paralelo. Essa combinação entre estas áreas da computação podem ajudar profissionais de vários segmentos da computação a resolver certos problemas de maneira em que o processamento possa ser dividido e distribuído pelas máquinas que formam um sistema, dessa maneira podem ser diminuídos os custos com hardware, (pois interligar máquinas caseiras em uma rede de alta velocidade é mais barato que a aquisição de um supercomputador), e possivelmente obter um melhor desempenho no processamento.

Para implementação de um sistema distribuído é necessário que haja uma maneira em que os computadores interligados se comuniquem pela rede. Existem alguns métodos utilizados na computação como *Sockets*, *RPC*, entre outros, porém existe um método exclusivo para a programação distribuída que utiliza a linguagem Java, o RMI *(Remote Method Invocated).*

O RMI vai proporcionar todas as ferramentas necessárias para a implementação de um sistema distribuído utilizando a linguagem Java. Como esta API faz a função de middleware, o programador fica despreocupado com diferenças entre tecnologias de rede e de hardware por exemplo.

A comunicação então é feita pela invocação de métodos remotos, onde um cliente utiliza os serviços disponibilizados pelo servidor.

Ainda mais interessante é utilizar o Java RMI como método de comunicação em uma rede com vários computadores para solucionar determinado algoritmo ou resolver algum problema computacional. Pois este é o objetivo, após o término do processamento, surge a possibilidade de ser feita uma análise entre dois ambientes de execução diferentes, em um ambiente utilizando RMI como método de comunicação e em outro ambiente utilizando outro método de comunicação, onde será possível identificar resultados em relação ao desempenho, e verificar em qual momento é mais vantajosa à utilização do Java RMI.

II. SISTEMAS DISTRIBUIDOS

O agrupamento de vários computadores que estão interligados por meio de uma rede de alta velocidade por onde podem se comunicar é denominado sistema distribuído [2]. O principal objetivo que leva a construção de tal sistema é a necessidade dos usuários em compartilhar recursos sendo que o maior exemplo de um sistema distribuído é a internet, que comunica e disponibiliza recursos para usuários do mundo todo. [1]

Sabendo disso, é possível identificar algumas características presentes em um sistema distribuído, como: falhas de componentes individuais, concorrência e inexistência de um relógio global. Por ter estas características, existem alguns desafios que estão no caminho da implementação de um sistema distribuído, são eles: [1] [2].

- Transparência: levar ao usuário a percepção de estar utilizando um sistema único e coerente.
- Escalabilidade: o sistema tem condições de ser expandido minimizando a perda de desempenho.
- Confiabilidade: proporcionar certa tolerância a falhas, segurança sobre recursos e disponibilidade do sistema.
- Flexibilidade: a facilidade em se realizar mudanças seguindo padrões pré-estabelecidos.

A presença de computadores e tecnologias diferentes em um sistema distribuído traz a preocupação em relevar as diferenças, para que haja uma colaboração entre os componentes do sistema na troca de informação e também facilitar ao desenvolvedor a programação do sistema. Esta responsabilidade é de uma camada denominada middleware que fica situada entre a camada de aplicação e as camadas de hardware e sistema operacional. A Fig. 1 apresenta o posicionamento da camada de middleware em um sistema distribuído, com ela os usuários podem interagir de maneira consistente e uniforme.

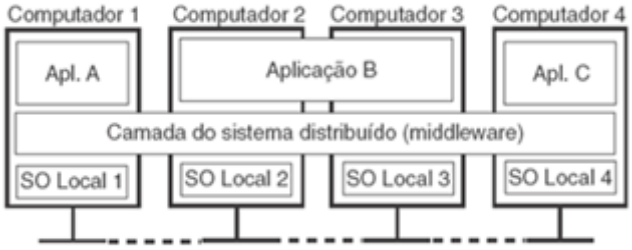

Figura 1. Camada de middleware em um sistema distribuído [2].

A construção de um sistema distribuído além de ter que superar estes desafios, também deve ter uma finalidade bem definida. Dentre as finalidades de um sistema distribuído estão: sistemas para fins computacionais, sistemas para disponibilizar informação e sistemas embutidos. [2]

O foco deste trabalho são os sistemas distribuídos para fins computacionais. Este tipo de sistema pode ter uma ramificação para dois tipos de sistemas distribuídos computacionais, os que possuem hardwares semelhantes e que estão conectados por uma rede de alta velocidade, chamado de computação em cluster, e os que possuem hardware, sistema operacional, tecnologia de rede diferentes, e ainda podem estar situados em domínios administrativos diferentes, esta é a computação em grade [2].

Com a proposta de verificar e analisar o desempenho de determinado algoritmo o mais apropriado a se implementar são os sistemas de computação em cluster. Estes possuem características mais apropriadas para se distribuir funções em um ambiente onde o desempenho é um fator muito importante. Na maioria das vezes a computação em cluster é designada a resolver problemas complexos que são divididos em problemas menores e de menor complexidade entre os nós que fazem parte do sistema, estas partes têm a possibilidade de serem executadas em paralelo, utilizando cos conceitos da computação paralela [3].

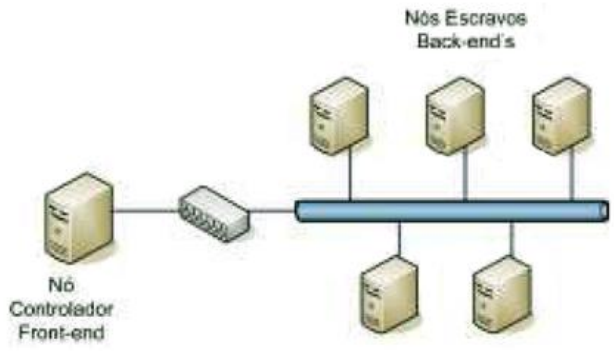

Figura 2. Arquitetura em cluster [3].

A arquitetura em cluster que é apresentada na Fig. 2 tem características que trazem vantagens na utilização deste modelo, são estas:

- Expansibilidade: o poder de processamento do cluster pode ser aumentado apenas adicionando um novo computador ao cluster, sem que haja modificações relevantes no sistema.
- Tolerância a falhas: um problema que afeta um nó do cluster, não interfere no resto do mesmo, o nó com problema é desabilitado e um balanceamento da carga de processamento é feito entre os nós ativos.
- Baixo custo: clusters podem ser montados utilizando máquinas convencionais, não se limitando a apenas um fornecedor, o que traz uma viabilidade financeira no cenário atual.

Como tudo tem suas vantagens e desvantagens, as desvantagens dos sistemas em clusters são:

- A manutenção dos componentes: os componentes devem estar em perfeita condições, e fazer esta manutenção em um cluster muito grande pode ser um problema.
- Gargalo de informações: a rede utilizada para troca de informações se torna um gargalo, já que sua velocidade é consideravelmente menor a de um barramento de dados.

Atualmente, a computação em cluster é encontrada em ambientes que trabalham com processamento de dados, computação cientifica, previsões meteorológicas, análises financeiras, entre outras, e a característica semelhante que está presente nessas áreas de aplicação é que todas exigem um alto grau computacional [3].

Um ponto importante em sistemas distribuídos é a comunicação entre os computadores da rede. Por serem fracamente acoplados a comunicação entre os computadores é feita através de troca de mensagens que utiliza o modelo requisição-resposta (bloqueantes ou não-bloqueantes), porém existem diversas maneiras de se efetuar esta troca de mensagens, RPC (Remote Procedure Call), RMI (Remote Method Invocation), Corba (Common Object Request Architecture) são algumas destas maneiras [1] [2].

III. JAVA RMI

O RMI é uma interface da linguagem Java que permite que aplicações distribuídas sejam criadas, apenas com a utilização da linguagem Java. Esta interface atua como middleware, que abstrai ao programador questões envolvendo diferenças em hardware, rede, entre outros aspectos que poderiam levar preocupações desnecessárias ao programador que deverá concentrar seu trabalho apenas na questão da programação [4].

Esta interface se posiciona em uma camada do middleware que fica abaixo da camada de aplicação e acima das camadas de protocolo de comunicação, sistema operacional e hardware, como mostrado na Fig. 3. O RMI abstrai ao programador as diferenças encontradas nas camadas inferiores [1].

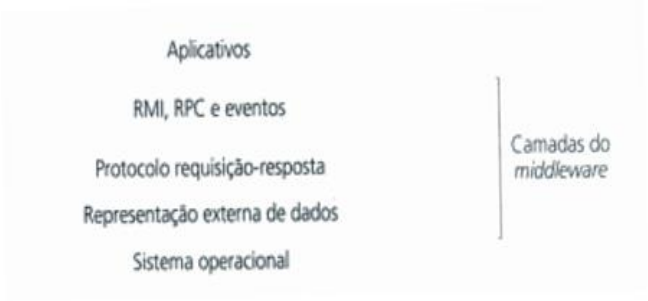

Figura 3 – Camadas do Middleware [1].

O programador não terá muita dificuldade em implementar um sistema com objetos distribuídos, pois o RMI possui como uma de suas vantagens a facilidade em se programar em distribuído, pois mantém uma semântica muito semelhante a programação que envolve objeto locais. A programação só não é idêntica a quando programamos com objetos locais, pois em sistemas distribuídos há questões de rede envolvida que vão fazer com que seja necessário o tratamento de exceções e execuções particulares. Uma característica do RMI já citada anteriormente que pode ser considerada como uma desvantagem, é que ele deixa a programação restrita à linguagem Java [4].

Um sistema distribuído implementado utilizando a linguagem Java trabalhará com o conceito de orientação a objetos. Com isso uma classe irá conter todos os serviços disponibilizados pelo sistema, que ficarão disponíveis para invocações de clientes por meio de objetos distribuídos.

No momento em que o cliente se vincula a um objeto distribuído acontece a criação de duas entidades. No lado do cliente é criado um proxy denominado stub, que fica situado no espaço de endereçamento do cliente. O stub é um representante local que implementa a mesma interface que o objeto que está representando, sua responsabilidade é fazer a serialização dos parâmetros de uma invocação e envia-la ao servidor. No lado do servidor um objeto denominado skeleton faz a comunicação direta com o cliente por meio do stub, o skeleton tem a responsabilidade de decodificar os parâmetros recebidos na invocação e repassar ao servidor para que o método mais apropriado para aquela invocação seja executado, e mais tarde, serializar a resposta que será encaminhada ao stub [4].

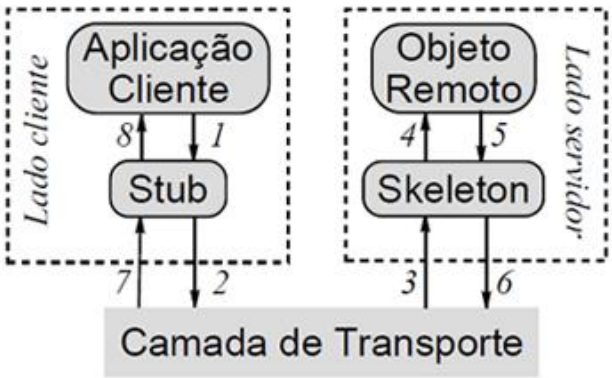

Figura 4. Estrutura de um cliente e de um servidor na comunicação utilizando Java RMI [4].

A estrutura de uma máquina cliente e de uma máquina servidor de um sistema distribuído que utiliza RMI na comunicação está bem representada na Fig. 4, com esta imagem também é possível identificar a localização do stub e do skeleton e a sequência de ações que acontece quando uma invocação é feita. Esta sequencia é composta pelas seguintes etapas:

- Surge a necessidade da aplicação cliente utilizar um serviço disponibilizado pelo servidor. Esta necessidade é passada ao stub que ficará responsável por toda a comunicação e decodificação dos parâmetros envolvidos.
- O stub inicia uma conexão com o servidor em que o objeto remoto está localizado. As informações que precisam ser enviadas ao servidor são colocadas do parâmetro da invocação e serializadas pelo stub para posterior envio. A invocação está pronta e então é transmitida ao servidor pela rede.
- O servidor cria um elemento denominado skeleton que ficará responsável por receber as invocações que chegam do stub.
- Logo após receber a invocação, o skeleton decodifica os parâmetros e repassa ao servidor para que este execute o método que irá atender a necessidade da aplicação cliente.
- O método é executado e o resultado é passado ao skeleton para que este faça a comunicação.
- Antes de enviar a resposta ao cliente, o skeleton serializa os parâmetros da resposta e então a transmite pela rede.
- O stub recebe a resposta enviada pelo skeleton.
- Após receber, o stub decodifica os parâmetros da mensagem de reposta e repassa a aplicação cliente para que esta utilize o resultado do serviço utilizado [4].

Todos os serviços oferecidos ficarão disponíveis em um objeto remoto como dito anteriormente, porém para que o cliente possa efetuar consultas aos serviços oferecidos é preciso que o objeto remoto se registre junto a um servidor. O objeto se registra por meio do método *bind* disponibilizado pela classe *Naming*, passando uma referência do objeto (stub) e o nome do mesmo. Este servidor registra os objetos remotos associando-os a um nome, é por meio deste nome que o cliente irá procurar o objeto, através do método *lookup* que também é disponibilizado pela classe *Naming*. Este serviço de localização de nomes é denominado RMIRegistry e ao final o mesmo retorna ao cliente a localização do objeto remoto [4] [1].

IV. COMPUTAÇÃO PARALELA

O surgimento da computação paralela se deu pela necessidade de se obter um desempenho que as arquiteturas sequências não conseguiam obter. Cada vez mais a computação exige um alto grau de complexidade e desempenho para resolver problemas que estão em torno de diversas áreas de atuação, como: áreas científicas, médicas, militares entre outras, necessitam de um alto poder computacional para fazer o processamento de algoritmos complexos com grande quantidade de dados [7].

"A computação paralela é o processamento de informações que enfatiza a manipulação concorrente dos dados. Que pertencem a um ou mais processos que objetivam resolver um único problema" [10].

O entendimento de computação paralela envolve o conhecimento de três essências, são elas: concorrência, paralelismo e granularidade [6].

A concorrência é a disputa entre dois ou mais processos para execução. O paralelismo consiste na divisão de uma aplicação em partes, onde essas partes são executadas por vários elementos de processamento no mesmo intervalo de tempo, buscando o melhor desempenho. Com isso, processos paralelos também são concorrentes, porém processos concorrentes nem sempre são paralelos [7]. A granularidade é definida pela razão entre o tempo necessário para o calculo de determinada operação e os custos envolvidos nas trocas de mensagens. A granularidade pode ser fina (conjunto de instruções simples), média ou grossa (conjunto de instruções complexas) [7].

No entanto uma máquina ou um sistema que trabalhe em paralelo não será de muita utilidade se não for extraído ao máximo seu poder de processamento, para que isto ocorra é preciso que se tenham também excelentes programas paralelos, onde se paralelize tudo o que é possível, o que é uma tarefa complexa para os programadores [7].

A. Arquiteturas Paralelas

As arquiteturas paralelas procuram agrupar unidades que possuem características semelhantes. Dentre as diversas classificações propostas, a taxonomia de Flynn é a mais citada, nela são formados quatro grupos para classificar as arquiteturas.

Levando em conta duas questões, o fluxo de instruções e o fluxo de dados, que podem ser definidos como simples ou múltiplos. As classes formadas foram: SISD (Single Instruction Single Data), MISD (Multiple Instruction Single Data), SIMD (Single Instruction Multiple Data), MIMD (Multiple Instruction Multiple Data [6].

A taxonomia de Tanenbaum subdividiu a classe MIMD em outras duas novas classes, a de multiprocessadores, que são processadores independentes que utilizam apenas um único espaço de endereçamento (fortemente acoplados) efetuando loads e stores (carrega e armazena) para a comunicação entre os processadores. E a de multicomputadores que são processadores que possuem um espaço de endereçamento distinto e que se comunicam através de troca de mensagem utilizando sends e receives (envia e recebe) através de uma rede de comunicação [6] [7].

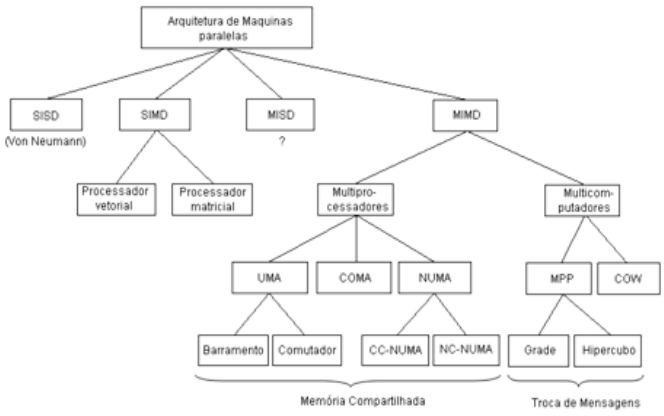

Figura 5 – Classificação das arquiteturas paralelas [6].

A Fig. 5 representa as várias classes de máquinas com arquitetura paralela que podem existir segundo a taxonomia de Tanenbaum.

B. Computação Paralela Distribuída

Duas ramificações da computação que juntas proporcionaram uma maneira viável economicamente de ganhar desempenho. A utilização de sistemas distribuídos trouxe as vantagens de transparência de acesso aos recursos, tolerância a falhas, e confiabilidade. Já a aplicação da computação paralela sobre tais sistemas trouxe a eficácia e o ganho de desempenho.

O alto custo das MPP's (*Massively Parallel Processing*) dificultava o uso do processamento paralelo, porém com o aperfeiçoamento das redes de comunicação e as máquinas populares se tornando máquinas com alto poder de processamento, os sistemas distribuídos se difundiram com a computação paralela proporcionando compartilhar recursos e ter um alto desempenho [6].

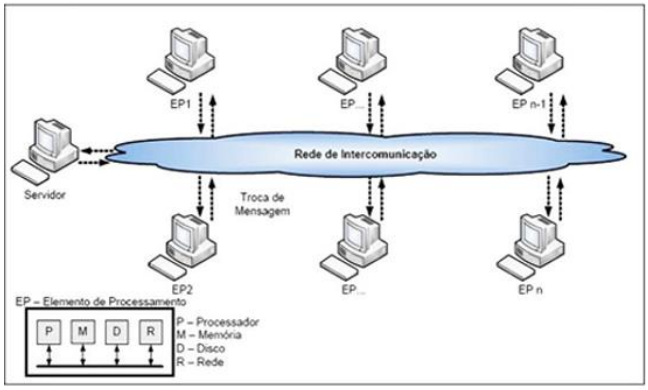

Figura 6 – Sistema Computacional Paralelo Distribuído [6].

C. Métodos para Análise

Algumas métricas são importantes para se medir os ganhos ou as perdas de desempenho e verificar a qualidade do programa paralelo perante uma aplicação qualquer. São elas: [7]

- *Speedup*: é obtido através do tempo gasto para execução do programa sequencial(ts) pelo tempo gasto para execução do programa em paralelo(tp).

$$S(p) = ts/tp \qquad (1)$$

- Eficiência: é obtida utilizando o s*peedup* pelo número de processadores utilizados na execução.

$$E(p) = Sp/p \qquad (2)$$

V. ALGORITMOS DE CRIPTOGRAFIA

A criptografia esteve sempre presente ao longo da história, e foi com a evolução dos computadores e com a necessidade de uma segurança cada vez mais forte que a criptografia entrou de vez como tema de diversas pesquisas cientificas.

Existem dois tipos de criptografia, a simétrica, e a assimétrica. Na criptografia simétrica, é utilizada uma mesma chave tanto para criptografar como para desencriptografar, este método é relativamente rápido, porém se torna inviável em transações que envolvam internet. O algoritmo mais utilizado hoje em dia para este tipo de criptografia é o AES [8].

Já na criptografia assimétrica são utilizadas duas chaves, uma para criptografar a mensagem e outra para desencriptografar. Neste tipo de criptografia temos o conceito de chave pública e chave privada. A chave pública é utilizada para criptografar a mensagem e pode ser de conhecimento de qualquer pessoa, enquanto a chave privada é utilizada para desencriptografar a mensagem, esta chave é de conhecimento exclusivo do destinatário e ela deve ser referente a uma única chave pública, pois apenas ela conseguirá decifrar a mensagem. Quando uma pessoa A deseja enviar uma mensagem para pessoa B, a pessoa A deve utilizar a chave pública de B para criptografar a mensagem, e depois de enviada, a pessoa B usa sua chave privada para decifrar a mensagem. Este tipo de criptografia é mais lenta que a simétrica e o algoritmo mais utilizado hoje em dia é o RSA [8] [9].

VI. CONCLUSÕES PARCIAIS

Com todas as referencias bibliográficas coletadas e os capítulos de interesse separados para posterior leitura, a monografia se encontra em fase de estudo e análise dos sub temas que compõem o tema principal, como computação paralela, sistemas distribuídos, RMI e algoritmos de criptografia.

Tomando como base os estudos feitos sobre as referências bibliográficas pude concluir até o momento que o principal desafio neste projeto será paralelizar ao máximo o algoritmo criptográfico para que se possa obter o melhor desempenho. A utilização de Java RMI para comunicação torna fácil a programação, pois se assemelha muito a programação sequencial, por outro lado a tarefa de distribuir as partes do algoritmo entre as máquinas que compõem o sistema será outra questão que vai interferir diretamente no desempenho. Espero ao final da implementação, ao serem feitas todas as análises, concluir se realmente é vantajoso executar um algoritmo em paralelo utilizando RMI e poder mostrar exatamente qual o ganho ou qual a perda de desempenho na execução. Com isso contribuir para que futuros projetos que irão utilizar algoritmos paralelos com RMI ou até mesmo projetos que terão apenas parte de seu todo inseridos neste contexto possam se referenciar nesta monografia para levar informações que irão agregar aos respectivos trabalhos.

REFERÊNCIAS

[1] G. Coulouris, J. Dollimore and T. Kindberg, “Sistemas Distribuídos”, 4 ed., São Paulo: Artmed, 2007.

[2] A. S. Tanenbaum, M. V. Steen, “Sistemas Distribuidos”, 2 ed., Rio de Janeiro: Prentice-Hall, 2007.

[3] H. V. Bacellar, “Cluster: Computação de Alto Desempenho”, Campinas: Instituto de Computação, 2010.

[4] F. M. Q. Pereira, “Arcademis: Um Arcabouço para Construção de Sistemas de Objetos Distribuídos em Java”, Belo Horizonete: UFMG Dezembro 2003.

[5] R. J. Sabatine, “Implementação de Algoritmo Paralelo para Apoiar o Processamento de Imagens utilizando JPVM”, Marilia: Univem, 2007, pp. 23-43.

[6] P. T. M. Saito, “Otimização do Processamento de Imagens Médicas Utilizando a Computação Paralela”, Marilia: Univem, 2007, pp. 22-62.

[7] [R. C. Detomini, “Explroração de Paralelismo em Criptografia utilizandos GPUs”. S. J. R. Preto, Universidade Julio de Mesquita Filho, 2010.

[8] E. F. F. Bernardi “Utilização de programação paralela e distribuída para quebra de chaves de criptografia RSA”. Porto Alegre: PUCRS.

[9] M. J. Quinn. “Designing Efficient Algorithms for Parallel Computers”. McGraw-Hill, Inc., Nova Iorque, NY, EUA, 1987.