UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PLATAFORMA COMPUTACIONAL HÍBRIDA DE COPROCESSAMENTO PARALELO DISTRIBUÍDO POR WEB SERVICES APLICADA À RADIO-INTERFEROMETRIA

GUSTAVO POLI LAMEIRÃO DA SILVA

ORIENTADOR: PROF. DR. JOSÉ HIROKI SAITO

São Carlos – SP

Agosto/2013

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

PLATAFORMA COMPUTACIONAL HÍBRIDA DE COPROCESSAMENTO PARALELO DISTRIBUÍDO POR WEB SERVICES APLICADA À RADIO-INTERFEROMETRIA

GUSTAVO POLI LAMEIRÃO DA SILVA

Tese apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação, área de concentração: Processamento de Imagens e Sinais

Orientador: Prof. Dr. José Hiroki Saito

São Carlos – SP

Agosto/2013

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária/UFSCar

S586pc

Silva, Gustavo Poli Lameirão da.
Plataforma computacional híbrida de coprocessamento paralelo distribuído por web services aplicada à radio-interferometria / Gustavo Poli Lameirão Da Silva. -- São Carlos : UFSCar, 2013.
144 f.

Tese (Doutorado) -- Universidade Federal de São Carlos, 2013.

1. Arquitetura de computador. 2. GPU. 3. CUDA. 4. Rádio-interferometria. 5. Correlação de sinais complexos. I. Título.

CDD: 004.22 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

"Plataforma Computacional Híbrida de Coprocessamento Paralelo Distribuído por Web Services Aplicada à Rádio-Interferometria"

Gustavo Poli Lameirão da Silva

Tese de Doutorado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Doutor em Ciência da Computação

Membros da Banca:

Prof. Dr. José Hiroki Saito
(Orientador – PPG-CC/UFSCar)

Prof. Dr. Márcio Merino Fernandes
(DC/UFSCar)

Prof. Dr. Célio Estevan Moron
(DC/UFSCar)

Prof. Dr. Edward David Moreno Ordonez
(UFS)

Prof. Dr. Stephan Stephany
(INPE)

São Carlos
Agosto/2013

A meus Pais Edmaldo e Judith, a minhas filhas Luísa e Manoella e a minha amada esposa Giuliana.

AGRADECIMENTOS

A Deus por me conceder a graça da vida, saúde e amigos.

Ao Prof. Saito, que neste trabalho foi em particular mais que um orientador, agindo como um pai, tendo me chamado a atenção na hora certa para que eu pudesse voltar a razão e concluir o Doutorado.

Ao Prof. Dr. José Roberto Cecatto do INPE (Instituto Nacional de Pesquisas Espaciais) pela atenção a este trabalho, auxiliando-me nos ensinamentos à rádio-interferometria.

À Prof. Dra. Maria do Carmo do Departamento de Computação da UFSCar que dedicou seu tempo tempo na correção do artigo de reconhecimento de explosões solares, que foi uma das aplicações implementadas utilizando a plataforma criada neste trabalho.

Ao Dr. James Peters da Un. Manitoba e Dra Sheela Romanna, da Un. de Winnipeg no Canada, pela parceria no trabalho publicado em revista.

À minha esposa Giuliana, companheira presente e amada, pelo carinho e atenção que me permitiram dedicar de forma exclusiva a esta jornada.

Às minhas filhas Luísa e Manoella, fontes de impiração e motivação.

Aos meus pais pela motivação e ajuda que com certeza sem esta eu não teria conseguido terminar o Doutorado.

Aos meus sogros Cleuza e Sebastiano que inúmeras vezes foram porto de descanço e boas companhias para "recarregar as baterias" e começar com vigor uma nova semana.

Aos grandes amigos Pedrinho e Marcia pelas ótimas rizadas que sempre me fizeram lembrar para que a ciência existe.

À CAPES pela bolsa de estudos durante a fase final do Doutorado.

Aos meus colegas de laboratório pelo ambiente agradável de trabalho e pela colaboração indireta, mas importante por meio de discussões bem humoradas no momento do café.

Deixa que a tua coragem se meça com as dificuldades que surjam no caminho. Não há vontade sem resistência.

N. Sri Ram

RESUMO

Os requisitos impostos pelas novas aplicações, sejam estas científicas, ou não, apresentam grandes desafios à computação. Não existe uma arquitetura de computadores "perfeita" que seja capaz de atender a todos estes requisitos. A configuração de arranjos paralelos e híbridos de computadores se apresenta como uma solução para este cenário, ou seja, a configuração de arranjos de pares CPU-Coprocessador, pode ser especializada para o processamento de uma aplicação distintas. Este trabalho de doutorado propõe uma plataforma computacional paralela e híbrida distribuída denominada CoP-WS, que utiliza a tecnologia de interoperabilidade conhecida como Web Services. Como coprocessador é utilizada a unidade de processamento gráfico conhecida como GPU, cuja função tem sido de processamento paralelo ao nível de threads, para aplicações gerais nos últimos tempos. A prova de viabilidade da plataforma implementada foi inspirada na radioastronomia, tendo sido implementados dois aplicativos: um correlacionador complexo de sinais provindos dos arranjos interferométricos e um sistema para o reconhecimento de explosões solares, numa imagem de radiointerferometria solar. Ambos os processamentos podem ser inseridos num contexto de execução em pipeline, usando uma configuração suficiente de pares CPU-GPU, tendo de um lado a entrada dos sinais das antenas do arranjo interferométrico e do outro lado o resultado do processamento de reconhecimento de explosões solares. Em ambas aplicações os resultados foram satisfatórios sendo que no caso do correlacionador o tempo médio de processamento de cada ciclo de integração foi de aproximadamente 160 ms, e para a aplicação de reconhecimento de explosões solares, de 48 ms por imagem de disco solar.

Palavras-chave: GPU, CUDA, Arquitetura Paralela e Híbrida, Web Service, Correlacionador de Sinais, Reconhecimento de Explosões Solares

ABSTRACT

The requirements imposed by the new applications presents great challenges to the computation. There is not a perfect computer architecture, capable to attend to all the requirements. The parallel and hybrid computer arrangement rise as a solution to this scenario i.e., the CPU-Coprocessor pair arrangement can form a specialized computerized instrument for a special application task. This doctoral thesis proposes a parallel and hybrid computational platform denoted CoP-WS, that uses the interoperability technology known as Web Services. As coprocessor it is used the graphic processing unit, known as the GPU, functioning recently as parallel thread level processing of general use applications. The platform test of feasibility was inspired in radio astronomy, and it has been implemented two applications: a complex correlator of signals provided by a radio interferometric arrangement, and a flare recognition system with a solar radio interferometer image. Both processings can be inserted in the context of pipeline execution, using sufficient configuration of CPU-GPU pairs, having on one side the interferometric arrangement antenna signal input and in the other side the result of the solar flare recognition. The obtained results of the both applications show the feasibility of the CoP-WS platform, for greater volume of data being processed in quasi real time. In the case of the correlator the average processing time in each integration period was around 160 ms, and in the case of the solar flare recognition, 48 ms for each solar disk image.

Keywords: GPU, CUDA, Architecture Parallel Hybrid, Web Services, Sinal Correlator, Solar Flare Recognition

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Neste capítulo é apresentada a contextualização na qual este trabalho de doutorado se encontra inserido. São discutidos os elementos de motivação, bem como as contribuições obtidas com o seu desenvolvimento. No final deste capítulo é apresentada a organização dos demais capítulos desta Tese com uma breve discussão sobre o conteúdo encontrado em cada um destes.

As comunidades científica e tecnológica sempre foram fontes geradoras de grandes desafios para a área da computação. Os problemas apresentados por estas comunidades possuem características que os tornam particulares, como o grau de complexidade dos algorítmos a serem implementados; o grande volume de dados, que podem ultrapassar com facilidade a ordem de Terabytes; e o tempo de resposta cada vez mais curto, que em muitos casos se associam ao processamento em tempo real. Desta forma, a computação pode ser vista como o ponto de convergência destes desafios impostos pelas comunidades acima. Desafios estes que buscam a criação de novas tecnologias, de forma a permitir a continuidade de pesquisas atualmente existentes, e/ou a criação de novas áreas de pesquisa.

O homem depende naturalmente de sua relação com o Planeta Terra, e da relação desta com o Universo. As estações do ano, que indicam a posição do nosso planeta em sua órbita em torno do Sol, interfere diretamente na agricultura, e em outras atividades naturais como a pesca, ou a migração de animais. Contudo, todo este conhecimento do homem sobre o Universo estava condicionado a suas observações, em um primeiro momento com seus próprios olhos, depois com instrumentos ópticos simples, que foram evoluindo ao longo do tempo. Mas sempre era necessário uma observação "direta" do objeto/fenômeno de estudo.

Apenas em 1931, Jansky, Engenheiro de Rádio dos laboratórios da Bell Telephone em Halmdel, Nova Jersey, que tinha sido alocado para trabalhar em um estudo sobre as interfe-

rências que ocorriam nas ligações telefônicas entre os EUA e a Europa, por acidente, percebeu uma interferência desconhecida em sua antena, construída especialmente para aquela atividade. Depois de algumas observações e análise, Jansky percebeu que a fonte rádio que estava sendo recebida pela antena, tinha como origem, uma fonte fora do planeta Terra. Era o inicio da Radioastronomia.

A radioastronomia busca entender o Universo por meio de medidas da radiação proveniente deste, os quais são utilizados para a criação de modelos. Os instrumentos utilizados para realizar essas medições são os radiotelescópios, instrumentos construídos segundo o comprimento de onda das fontes de rádio de interesse de estudo, e que possuem como uma de suas características mais importantes a capacidade de discriminar os sinais provenientes de diversas posições do espaço. A grande contribuição oferecida pela radioastronomia é justamente sua capacidade de observar o Universo, por meio das ondas de rádio que são emitidas pelos corpos, ou seja, é possível estudar aquilo que não se vê diretamente.

Entretanto a radioastronomia apresenta um ponto negativo. O instrumento utilizado para suas observações, o radiotelescópio, apresenta uma resolução angular (capacidade de diferenciar objetos) pobre, quando comparada com outros instrumentos ópticos. Isso se deve ao fato de que a resolução angular esteja associada à razão entre o comprimento de onda da fonte observada e o diâmetro do instrumento, no caso do radiotelescópio, sua antena. Esta razão exige que um radiotelescópio, para ter uma boa resolução angular, tenha uma antena de grande diâmetro, o que levou a construção de instrumentos como o de Arecibo, em Porto Rico, cujo diâmetros é de aproximadamente 300 metros (ARECIBO, 2011).

Antenas com diâmetros tão grandes geram um outro problema que é a parte mecânica utilizada para fazer o rastreio da fonte de interesse. No caso de Arecibo este não possui tal parte mecânica, ficando a sua rotação condicionada à rotação do planeta. Ou seja, antenas com grande diâmetro são pesadas, e dificultam, ou até mesmo inviabilizam, a instalação de mecanismos que permitam o rastreio de fontes cósmicas. A solução para este problema está na construção de um radiointerferômetro.

Um radiointerferômetro usa o efeito da combinação dos sinais de duas ou mais antenas observando a mesma fonte de rádio, como se fossem uma única antena. A disposição das antenas, que formam o radiointerferômetro, é chamada de arranjo interferométrico, e a distância entre cada par de antenas, linha de base. Como o arranjo interferométrico se comporta como uma única antena, a combinação dos sinais dessas antenas pode ser vista como um único sinal gerado por uma única grande antena virtual. O diâmetro desta antena virtual é a maior linha de base do arranjo, e o número de antenas do arranjo garante uma melhor cobertura do plano de

observação.

A radioastronomia solar é uma parte da astronomia que se dedica ao estudo da radiação eletromagnética proveniente de fenômenos solares, os quais propagam através do espaço em direção à Terra. Estes estudos permitem uma melhor compreensão dos processos físicos solares e das relações entre a atividade solar e a atividade interplanetária subsequente (FERNANDES, 1997). Dois destes fenômenos são de grande importância e estão relacionados com os ciclos de máximos solares, que são: os *flares* e CMEs (*Coronal Mass Ejection* - Ejeção de Massa Coronal) (CECATTO, 2003).

Os *flares* são fenômenos de natureza explosiva com duração que varia de alguns milissegundos até algumas horas, e que liberam uma grande quantidade de energia ($10^{23} - 10^{32}$ ergs) na forma de radiação em toda faixa do espectro eletromagnético, que vai desde os raios-γ até as ondas de rádio. A liberação de energia solar durante a ocorrência dos *flares* solares possui efeito catastrófico na magnetosfera e ionosfera terrestre, além de afetar diretamente instalações de alta tecnologia, como satélites, meios de comunicação e transmissão de energia, que são extremamente sensíveis à atividade solar. Tais efeitos estão também relacionados à expulsão de grandes quantidades de matéria pelo Sol, os CMEs. Estes são fenômenos de origem pouco conhecida, que ocasionam enormes arcos em erupção, que se expandem para o espaço interplanetário, e em grande parte, sua ocorrência está associada aos *flares*.

Dentro do contexto da radioastronomia solar, o INPE (Instituto Nacional de Pesquisas Espaciais) vem trabalhando na construção de instrumentos para o estudo destes eventos. Um de seus projetos é o radiointerferômetro BDA *(Brazilian Decimetric Array*), que atualmente, encontra-se na Fase II de seu desenvolvimento, com 26 antenas em seu arranjo, que proporciona uma resolução máxima de 44"x 68" de arco supondo que o receptor esteja operando em 5,6 GHz.

O volume de dados produzido por um radiointerferômetro, pode inviabilizar seu armazenamento para posterior processamento, tornando conveniente que a correlação destes sinais seja realizada em tempo real. Assumindo grandes arranjos de antenas, o poder computacional exigido para este processamento só poderia ser alcançado por meio da utilização de circuitos eletrônicos dedicados.

Com a evolução do poder computacional e com o desenvolvimento de novos instrumentos de radioastronomia, a utilização de *clusters* de computadores se torna uma opção viável, não apenas por ser de menor custo, em relação ao uso de circuitos eletrônicos dedicados, como também pela flexibilidade e escalabilidade apresentada (WOODS, 2010). Dentre as tecnologias computacionais que evoluíram, e que podem ser utilizado para esta finalidade, pode-se destacar o microprocessador Cell da IBM, utilizado no *Playstation* III, onde um *cluster* de *Playstation*

III é utilizado pela equipe do DiFX (DELLER et al., 2007), um *software* para a correlação de sinais de radiointerferômetros.

Outra tecnologia que evoluiu em seu poder computacional é o GPU (*Graphic Processing Unit*). Até o final da 1990 o GPU era visto como uma placa gráfica, um dispositivo para auxiliar na visualização de imagens. Impulsionado, principalmente, pela indústria de jogos, os GPUs passaram a ter processadores vetoriais especializados no processamento de imagens, o que lhes forneceu um grande poder computacional, chamando a atenção da comunidade científica, que interceptando o *pipeline* gráfico da placa, inseriram instruções para processamentos não gráficos, ou seja, para processamento genérico. Em 2006, a NVIDIA lançou o CUDA (*Compute Unified Device Architecture*), uma arquitetura híbrida (*hardware* e *software*), que permite que aplicações sejam executadas diretamente no GPU, sem a necessidade da intervenção no *pipeline* gráfico. Desta forma, o GPU passou a ser visto, não mais como uma placa gráfica, mas como um dispositivo de processamento maciçamente paralelo.

Com o CUDA, a visão do GPU como um dispositivo de apoio ao processamento do CPU fica mais nítido, fornecendo à comunidade científica uma ferramenta de desenvolvimento de aplicativos genéricos simples e eficiente. Dentro da radioastronomia, o GPU já tem sido testado quanto a sua viabilidade de uso no processamento de correlação dos sinais de arranjos interferométricos. Christopher John Harris, em 2009(HARRIS, 2009), fez uma proposta de um correlacionador baseado em GPU, e em 2010, Andrew Woods (WOODS, 2010), fez uma evolução no trabalho iniciado por Harris, além de comparar o correlacionador implementado em um CPU, GPU e FPGA (*Field-Programmable Gate Array*), e seguindo esta linha evolutiva, este trabalho apresenta uma alternativa do emprego desta tecnologia utilizando um arranjo de nós computacionais reconfigurável por meio de Web Services, o que permite o processamento de um volume de dados maior do que apenas o de um único par CPU-GPU.

E é dentro deste contexto, das necessidades de processamento de dados da radioastronomia que se encontra inserido este trabalho de doutorado, onde, por meio da implementação de uma plataforma computacional híbrida e distribuída, buscou-se configurar um arranjo computacional que pudesse ter seus elementos especializados, segundo as necessidades/características das rotinas as quais seriam responsáveis pelo processamento. Pela característica híbrida da plataforma implementada, optou-se por utilizar WS (*Web Services*) como interface de comunicação entre os elementos participantes do arranjo computacional, um WS é uma ferramenta computacional projetada para suportar a interoperabilidade, na interação entre máquinas. O WS, funciona como uma espécie de API (*Aplication Programming Interface*), abstraindo a implementação disponibilizada, por meio da chamada de seu serviço. Os detalhes sobre esta ferramenta são

descritos na Seção 2.10.

1.1 Trabalhos Relacionados

Neste trabalho de doutorado foi implementada uma plataforma computacional híbrida, de alto desempenho, que tem como foco o processamento de fluxo de grandes volumes de dados, dentro do escopo apresentado pela radioastronomia. Nesta seção é feito o posicionamento deste trabalho junto a outras pesquisas realizadas pela comunidade. São discutidas questões como a, criação de arranjos computacionais híbridos, o uso do GPU com a tecnologia de WS, e também sobre as duas aplicações que foram implementadas como prova de viabilidade: o reconhecimento de explosões solares e a correlação de sinais de arranjos interfeométricos via software.

1.1.1 Arranjos computacionais híbridos

A configuração de arranjos computacionais com diferentes tipos de tecnologia, organizados para trabalhar de forma colaborativa já é vista como uma necessidade, diante dos novos desafios impostos pelas aplicações. Mars e seus colaboradores em sua pesquisa analisando o comportamento de aplicativos em diferentes plataformas computacionais (MARS; TANG; HUNDT, 2011) , percebeu que existe uma diferença relativa no desempenho dos processadores, ou seja, o mesmo processador apresenta um desempenho diferente, quando executa diferentes aplicativos. Isso pode ser visto na Figura 1.1, onde 22 aplicativos são executados em plataformas Core i7, Core 2 e Pheonm X4, e a diferença de desempenho é mostrada em porcentagem de piora (*Slowdown*).

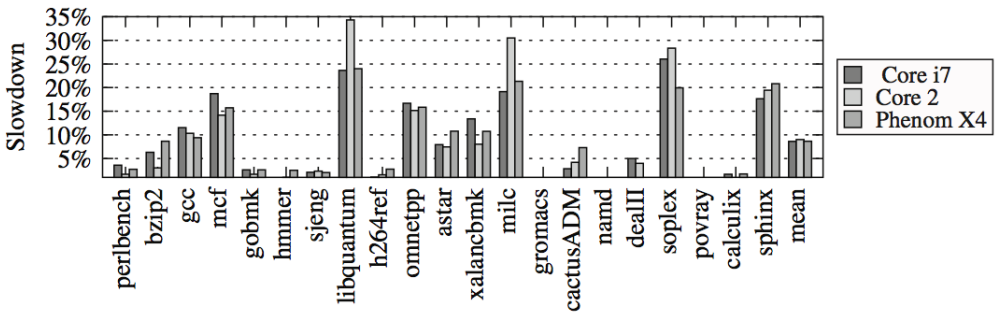


Figura 1.1: Diferença de desempenho de aplicativos em diferentes plataformas (MARS; TANG; HUNDT, 2011).

O cenário apontado por Mars, sugere que para se obter um melhor desempenho, deve-se conhecer não apenas uma arquitetura computacional, mas também o perfil da aplicação, ou

rotina, que deverá ser processada de forma a buscar uma melhor configuração da plataforma final. Jones e seus colaboradores fizeram um estudo comparando o GPU com a FPGA (JONES et al., 2010) e, durante este trabalho foram observados os desempenhos apresentados por cada uma dessas tecnologias comparadas com o perfil das aplicações que foram processadas em cada uma das plataformas.

Em 2009 Jamsek e seus colaboradores, trabalharam na configuração de um cluster híbrido, formado por GPU, CPUs e IBM Cell (JAMSEK; VANHENSBERGEN, 2009), utilizando como interface de programação a biblioteca MPI (*Message Passing Interface*), para realizar a troca de mensagens entre as estações do cluster e o CUDA 2.2 para a implementação das rotinas a serem processadas nos GPUs, em um ambiente Linux utilizando a distribuição RedHeat versão 5.2. Jamsek trabalhou na "fragmentação" do código onde cada rotina da aplicação poderia ser processada em uma plataforma diferente, já na tentativa de criar um *pipeline* entre as diferentes arquiteturas. Contudo, havia uma dificuldade referente à comunicação entre as diferentes aplicações implementadas para cada uma das arquiteturas.

Já em 2010, Tsoi e seus colaboradores trabalharam em uma idéia parecida com a de Jamsek, na criação de um cluster híbrido, mas Tsoi utilizou como diferentes arquiteturas o GPU, CPU e a FPGA (TSOI; LUK, 2010). Como pode ser observado na Figura 1.2, cada um dos nós do cluster configurado por Tsoi, era uma combinação de um CPU, GPU e um FPGA. Tsoi também utilizou como modelo de programação o MapReduce que tem foco no processamento de grandes columes de dados em ambiente distribuído, baseado em dois estágios denotados *Map* e *Reduce* (Seção 2.11), de forma a ter um melhor controle do processamento e da utilização de cada um dos dispositivos utilizados.

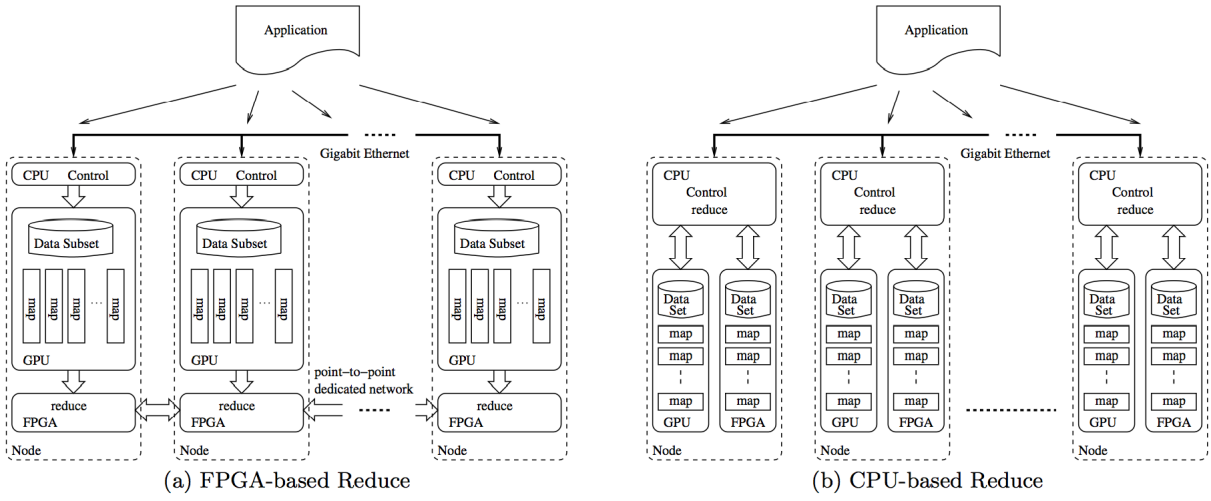


Figura 1.2: Arquitetura do cluster híbrido de CPU, GPU e FPGA implementado por Tsoi (TSOI; LUK, 2010).

1.1.2 Acesso a GPU por meio de Web Service

O trabalho de Gorder em 2008, discute o problema de grandes volumes de dados, e apresenta a computação em nuvens como sendo uma possível solução a esta questão, descrevendo os desenvolvimentos de empresas como a Google e a IBM (GORDER, 2008). Gorder descreve os problemas enfrentados em áreas como Astronomia, Metereologia, Microbiologia, além de outras aplicações da Física, como dinâmica de fluídos e colisão de particulas , apresentado o GPU/CUDA como um dispositivo de processamento massivamente paralelo que pode ser visto como uma ferramenta de apoio a estes temas.

Em 2010 Peng e seus colaboradores utilizaram um GPU do tipo NVIDIA Tesla C1060 para a pesquisa de strings dentro de um conjunto de páginas web, obtendo um speedup 28x quando comparado com soluções baseadas apenas no processador i7 (PENG; CHEN; SHI, 2010).

Si e seus colaboradores trabalharam na utilização do GPU no processamento de pesquisas XML dentro do contexto do WS criando uma versão paralela e simplificada da linguagem XPath (SI et al., 2011). Dentro da linha seguida por Si, Hung e seus colaboradores trabalham com o GPU/CUDA dentro do contexto de segurança dos WS por meio de um sistema de classificação de pacotes (HUNG et al., 2011). Um dos primeiros trabalhos associando Web Service e GPU foi desenvolvido por Liang e de seus colaboradores, onde foi utilizado o WS como interface de serviço para a separação de sinais, rotina que era executada por uma GPU GeForce 9500 GT (LIANG et al., 2011). Savytskyi com seus colaboradores estenderam o conceito utilizado por Liang, onde o serviço estava sendo disponibilizado por apenas um computador, e trabalharam para uma interface baseada em WS para o MolDynGrid, um laboratório virtual dedicado ao estudo de dinâmica molecular (SAVYTSKI et al., 2011).

1.1.3 Reconhecimento de Explosões Solares

Em 2005, Qu e seus colaboratores, trabalharam em um projeto para reconhecimento de explosões solares de forma automática e em tempo real(QU et al., 2005). Neste trabalho foram comparados três técnicas para essa atividade: MLP - Multi-Layer Perceptron, RBF Radial Basis Function, e SVM Support Vector Machine. Foram utilizadas como imagens de entrada $H\alpha$ (Hydrogen-Alpha) que foram produzidas do instrumento *Big Bear Solar Observatory* na California. Este trabalho teve continuidade por Shih, em 2010, que processando imagem de 2032x2032 pixels, em um computador DELL Dimension L733r com um CPU de 733 Mhz obteve os seguintes resultados apresentados na Tabela 1.1.

Método	Taxa de Classificação	Tempo	
		Treinamento	Teste
MLP com 1000 interações	94.2%	60.20 s	0.01 s
RBF	95%	0.27 s	0.01 s
SVM	96.7%	0.16 s	0.03 s

Tabela 1.1: Resultados obtidos por Shih durante o processamento do reconhecimento de explosões solares (SHIH, 2010).

Em 2012, Piazzesi e seus colaboradores, utilizaram tanto o hardware, com camera CMOS (C-CAM BCi5), quanto o software (LabView) fornecidos pela empresa National Instruments para a construção de um sistema para reconhecimento de explosões solares cuja interface com o usuário pode ser vista na Figura 1.3 (PIAZZESI et al., 2012).

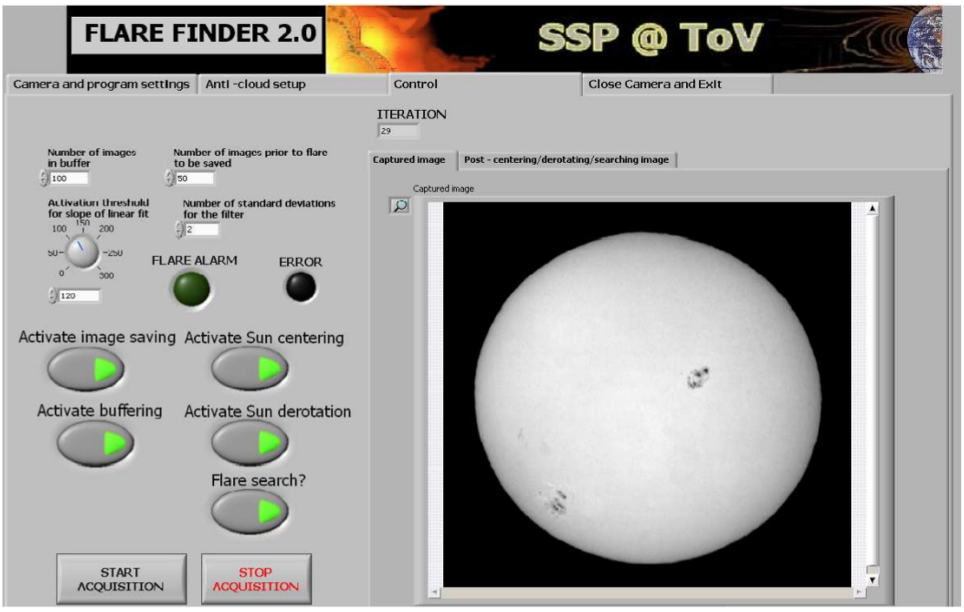


Figura 1.3: Interface com o usuário do sistema implementado por Piazzesi (PIAZZESI et al., 2012).

1.1.4 Correlação de Sinais Complexos de Arranjo Interferométrico via Software

Durante o processo de observação de uma fonte cósmica, um radiointerferômetro produz um grande volume de dados, que em muitos casos inviabiliza seu armazenamento, fazendo com que seja necessário seu processamento durante a aquisição destes. Essa característica faz com que uma implementação em *hardware* do processamento da correlação de dados de arranjos interferométricos seja normalmente utilizada. FPGA's (*Field Programmable Gate Array*),

que representam uma tecnologia de construção de circuitos eletrônicos reconfiguráveis, normalmente utilizada na implementação de protótipos, representam uma solução alternativa de implementação em *hardware*, para o processamento em tempo real.

A ideia de implementar um *software* para realizar a correlação dos sinais de um arranjo interferométrico não é nova e possui como uma das principais implementações, o DiFX (*Distributed FX*) que foi proposto e desenvolvida na Universidade de Swinburne (DELLER et al., 2007). Sua versão inicial foi para *cluster* de computadores, mas em 2009 recebeu uma nova versão para a arquitetura Cell da IBM (WAGNER; RITAKARI, 2009), tendo sido lançado a versão 2.0 em 2011 (DELLER et al., 2011), apresentando um ganho de desempenho significativo. A evolução do poder computacional dos GPUs, bem como a especialidade destes dispositivos para o processamento de grandes volumes de dados, chamaram a atenção da comunidade para a criação de uma arquitetura híbrida, de baixo custo conciliando a facilidade de manutenção e escalabilidade, acpectos que vem sendo discutida na comunidade astronômica há algum tempo.

Um dos pioneiros, dentro desta linha de pesquisa, é Harris (HARRIS, 2009), que apresentou uma proposta de correlacionador 100% baseado em GPU. No ano seguinte Woods (WOODS, 2010) otimizou a proposta de Harris por meio de uma arquitetura híbrida, utilizando CPU e o GPU, além de fazer uma comparação com arquiteturas mais tradicionais, como o FPGA.

1.2 Visão Geral e Contribuições

Como apresentado por Mars e seus colaboradores uma mesma aplicação pode apresentar um desempenho diferente em arquiteturas computacionais distintas (MARS; TANG; HUNDT, 2011). Na realidade cada tipo de problema possui suas próprias características, que são os elementos chave para a determinação da escolha de qual tipo de tecnologia pode ser utilizado de forma mais eficaz. Havendo ainda casos em que um mesmo problema, apresente em fases diferentes de seu processamento, a necessidade de mais de um tipo de tecnologia/arquitetura.

E é dentro desta linha de raciocínio que segue este trabalho de doutorado, focado na criação de uma plataforma computacional híbrida (PCH) que possa ser vista como uma opção viável para o processamento de grandes volumes de dados, produzidos por um fluxo contínuo, dentro de um tempo de resposta aceitável, segundo as especificações da aplicação a ser implementada, em diferentes tipos de domínios de problema, onde apresenta como principais requisitos:

1. Integração de diferentes tecnologias: a plataforma computacional deve ser vista como um ponto de convergência para o uso de diferentes tipos de tecnologias, servindo como

uma interface de comunicação entre estas, de forma a criar um ambiente de trabalho colaborativo;

2. Escalabilidade: a plataforma criada deve apresentar a capacidade de crescer de forma natural, ou seja, pela adição de novos nós computacionais com a menor intervenção quanto a requesitos de configuração de infra-estrutura;

3. Reconfigurável: diferentes tipos de problemas apresentam diferentes tipos de soluções, e a plataforma deve apresentar a capacidade de ser organizada, segundo seu fluxo de processamento, seguindo as necessidades do problema que se implementa;

4. Componentização: seguindo os conceitos de engenharia de software, a plataforma deve organizar um conjunto de recursos computacionais, que são especializados na execução de uma determinada tarefa, agindo como um "componente";

5. Reutilização do código: uma determinada tarefa pode ser implementada e configurada em um grupo de recursos computacionais, sendo que estes podem ser utilizados em mais de uma aplicação; e

6. Facilidade de Manutenção: a plataforma deve apresentar recursos que tornem simples, com o menor custo de processamento, a manutenção e/ou troca de elementos que compõe a mesma.

Desta forma a plataforma PCH implementada neste trabalho apresenta as seguintes características:

1. Ambiente distribuído: a plataforma é formada por um conjunto de computadores, chamados de nós computacionais, que trabalham de forma colaborativa, e que se encontram distribuídos em rede;

2. Incorpora o conceito de coprocessamento ao nó computacional: cada um dos nós computacionais pode possuir um dispositivo que funciona como um coprocessador, auxiliando o processador do nó na execução de tarefas nas quais este seja mais especializado;

3. Especialização do nó computacional na execução de tarefas: cada nó computacional é dedicado ao processamento de uma determinada aplicação, ou tarefa, de forma que cada nó possa ser visto estágio do fluxo de processamento dos dados;

4. Definição de um fluxo de processamento de dados: como cada nó computacional é especializado na execução de um determinado processo, os dados são enviados aos nós segundo

um fluxo de processamento, respeitando as fases de processamento do problema que se implementa. A saída dos dados do último nó deste fluxo, referente ao últmo estágio de processamento do problema implementado, é considerado o resultado do processamento, ou a saída da aplicação.

5. Processamento em *pipeline*: uma vez que os nós computacionais se encontram agrupados por bloco funcional, cada um destes blocos podem estar processando um conjunto de dados amostrados em um tempo diferente.

Com base nas características apresentadas, foram escolhidos como tecnologias a serem utilizadas na implementação da plataforma objeto deste trabalho:

1. Arquivos XML: a plataforma conta com um conjunto de arquivos de configuração que auxiliam na definição do fluxo de processamento, configuração dos nós do arranjo, entre outros. Estes arquivos de configuração são escritos em linguagem XML, descrito na Seção 2.9;

2. Web Services: esta tecnologia é utilizada como interface de comunicação entre os nós da plataforma, ou seja, cada nó computacional do arranjo representa um WS, onde os serviços disponibilizados representam as interfaces de chamadas para os serviços/tarefas implementadas neste nó; e o

3. GPU: como dispositivo de coprocessamento, nesta versão da plataforma, foi utilizado o GPU NVIDIA, e o ambiente de desenvolvimento CUDA, que descritos na Seção 2.12;

A configuração básica da plataforma PCH implementada neste trabalho pode ser resumida em um conjunto de estações de trabalho que são formadas por um par CPU-Coprocessador, contectados em uma rede TCP/IP, onde a interface de comunicação entre cada um destes é feita por meio do uso de WS, por estas características a plataforma PCH recebeu o nome de CoP-WS (Coprocessador Paralelo e Distribuído por Web Services). Por coprocessador deve ser entendido qualquer tipo de dispositivo que trabalhe de forma colaborativa com o CPU, e que se apresente como um opção mais adequada para o processamento de um determinado tipo de problema e/ou formatação de dados. Na versão atual do CoP-WS é utilizado o GPU como coprocessador, contudo, em casos particulares, este coprocessador poderia ser um FPGA, ou uma Balde Cell (GSCHWIND et al., 2005) por exemplo.

Para realizar a prova de viabilidade da plataform CoP-WS, foram implementados dois aplicativos: 1) correlacionador de sinais de arranjos interferométricos, como descrito na Seção

2.15; e 2) uma aplicação para a detecção de explosões solares utilizando o GLCM (*Gray-Level Co-ocuurence Matrix*) e a teoria de Near Sets, descritas nas Seções 2.14 e 2.13 respectivamente.

Com base no escopo deste trabalho e de seu projeto, são esperadas como principais contribuições da presente tese:

1. Configuração de uma plataforma de computação paralela híbrida, CoP-WS, escalável e de alto desempenho, para diversas aplicações;
2. Apresentação de solução alternativa de processamento de sinais de arranjos de radiointerferometrica, cujo desempenho é variável dependendo apenas do número de nós computacionais implementados;
3. Apresentação de uma solução genérica de sistema de computação para radio-interferometria solar em que todo o processamento desde a aquisição de sinais pelas antenas até o reconhecimento de flares seja possível.
4. Apresentação de uma arquitetura reconfigurável, baseada em software, com capacidade de processamento de dados em *pipeline* por meio da utilização do WS.

1.3 Organização dos capítulos

Esta Tese de Doutorado está organizada nos seguintes capítulos:

1. Introdução: este presente capítulo, responsável por apresentar o contexto deste trabalho, incluindo a sua referência junto a outros trabalhos realizados pela comunidade. Também são apresentadas as principais contribuições resultantes do desenvolvimento da tese;
2. Referência Bibliográfica: neste capítulo é realizada a descrição da pesquisa bibliográfica para o desenvolvimento deste trabalho. Este capítulo se encontra organizado em duas partes: 1) embasamento utilizado para o desenvolvimento da plataforma CoP-WS propriamente dito; e 2) embasamento utilizado para o desenvolvimento das aplicações para a prova de viabilidade e validação da proposta apresentada;
3. Descrição da Plataforma CoP-WS e Metodologia: capítulo onde é apresentada a metodologia seguida para o desenvolvimento da tese. Assim como no capítulo de Referência Bibliográfica, este também é organizado em duas partes, uma com a metodologia para o desenvolvimento da plataforma CoP-WS e a segunda para a metodologia para o desenvolvimento das aplicações de teste;

4. Resultados: neste capítulo são apresentados os resultados obtidos com o desenvolvimento da plataforma proposta, além dos resultados das aplicações implementadas; e

5. Conclusão: capítulo onde é realizada uma discussão sobre os resultados obtidos tanto pela plataforma quanto pelas aplicações de prova de viabilidade, e apresenta uma lista de sugestões para continuidade do desenvolvimento.

Capítulo 2

REFERENCIAL BIBLIOGRÁFICO

No capítulo anterior foi apresentada a introdução deste trabalho de doutorado, suas motivações, contribuições esperadas e a organização dos capítulos. Neste capítulo é apresentado o referencial bibliográfico utilizado para o desenvolvimento deste trabalho. Suas seções estão organizadas em duas partes: 1) teoria base para o desenvolvimento da plataforma CoP-WS propriamente dita; e 2) referência para a implementação das aplicações utilizadas para testar a viabilidade de uso da plataforma.

De forma resumida a plataforma implementada neste trabalho, chamada de CoP-WS, pode ser vista como sendo um conjunto de nós computacionais, onde cada um destes nós é formado por um par CPU-Coprocessador, onde o coprocessador pode ser qualquer tipo de dispositivo de processamento que venha a auxiliar o CPU na execução de sua tarefa. Uma característica da plataforma CoP-WS é que cada um destes nós possui consciência de seu papel dentro do fluxo de processamento de dados de uma determinada aplicação, e a troca de mensagens realizada entre estes nós é feita por meio da utilização de tecnologia para interoperabilidade conhecida como WS (*Web Services*). Apesar da plataforma permitir que seja utilizado qualquer tipo de dispositivo como coprocessador, a versão atual foi implementada utilizando apenas GPU's para esta finalidade.

2.1 Computação de Alto Desempenho

Visando atender a demanda crescente de processamento de alto desempenho, têm sido desenvolvidos vários modelos de arquiteturas paralelas. A maioria dessas arquiteturas é caracterizada pela presença de múltiplos processadores que cooperam entre si. No entanto, existem outros modelos de arquitetura paralelas, motivo pelo qual é feita a descrição geral seguinte desses modelos.

2.1.1 Arquiteturas Paralelas

Michael Flynn (FLYNN, 1972) classificou as arquiteturas paralelas de acordo com o fluxo de dados e instruções. Dependendo se estes fluxos são múltiplos ou não e por meio da combinação das possibilidades, Flimn propôs 4 categorias: SISD, SIMD, MISD e MIMD.

2.1.1.1 SISD

Essa categoria consiste na existência de um único fluxo de instruções e um único fluxo de dados, caracterizando uma arquitetura com um único processador, como mostra a Figura 2.1, onde se vê o fluxo de instruções partindo da memória M passando por uma unidade de controle (UC), para ser então processado (P) e tendo os dados sendo lidos ou gravados em uma posição de memória (M).

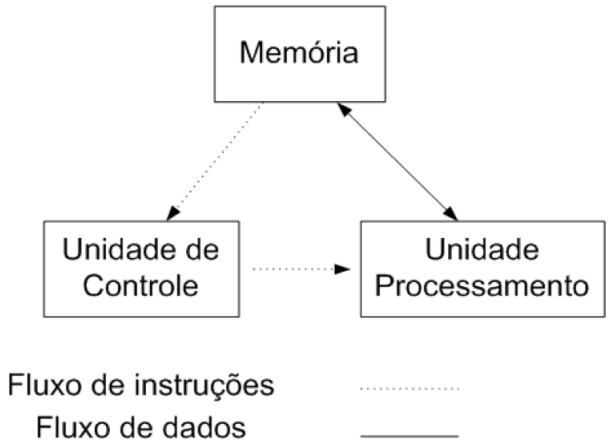


Figura 2.1: Diagrama de blocos de uma arquitetura SISD.

Um exemplo de fluxo de instruções a serem executadas nessa arquitetura pode ser observado na Figura 2.2, onde observa-se que a cada momento (tempo) existe apenas uma instrução sendo executada.

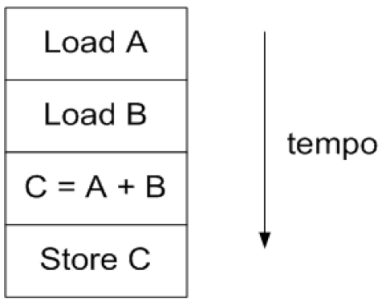


Figura 2.2: Fluxo de instruções sendo executadas na arquitetura SISD.

2.1.1.2 SIMD

Essa categoria consiste na existência de um fluxo único de instruções, porém que corresponde à manipulação de um fluxo múltiplo de dados sobre os quais se realizam operações idênticas em elementos de processamento distintos, que operam sincronizadamente entre si, como pode-se ver na Figura 2.3.

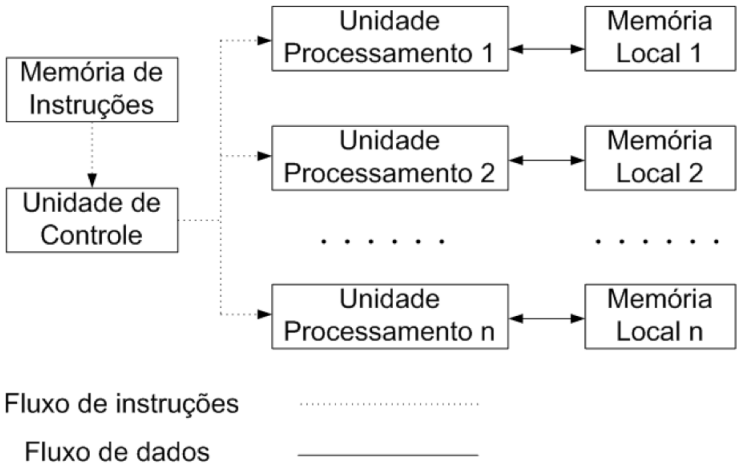


Figura 2.3: Diagrama de blocos de uma arquitetura SIMD.

Na Figura 2.4 pode-se ver um exemplo de um fluxo de instruções sendo executadas em diferentes processadores ao mesmo tempo.

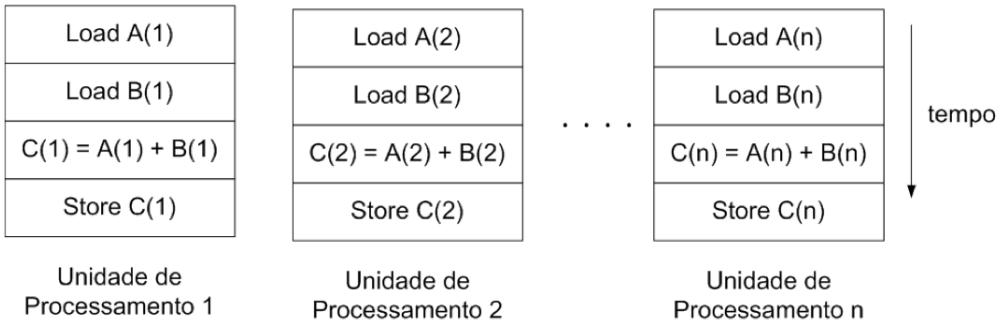


Figura 2.4: Exemplo de fluxo de instruções na arquitetura SIMD.

2.1.1.3 MISD

Essa categoria consiste na existência de fluxo múltiplo de instruções sobre fluxo único de dados, como pode ser observado na Figura 2.5.

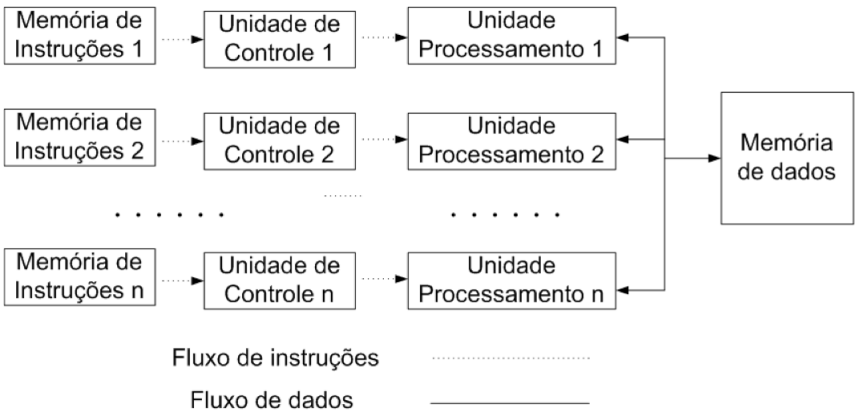


Figura 2.5: Diagrama de blocos de uma arquitetura MISD.

As máquinas com arquitetura MISD são classificadas diferentemente segundo vários autores. Para alguns, correspondem a processadores vetoriais; outros descrevem-nas como variantes de máquinas SIMD e para a maioria, porém, tais máquinas não têm existência prática.

2.1.1.4 MIMD

Essa categoria consiste na existência de fluxo múltiplo de instruções sobre fluxo múltiplo de dados como pode ser observado na Figura 2.6.

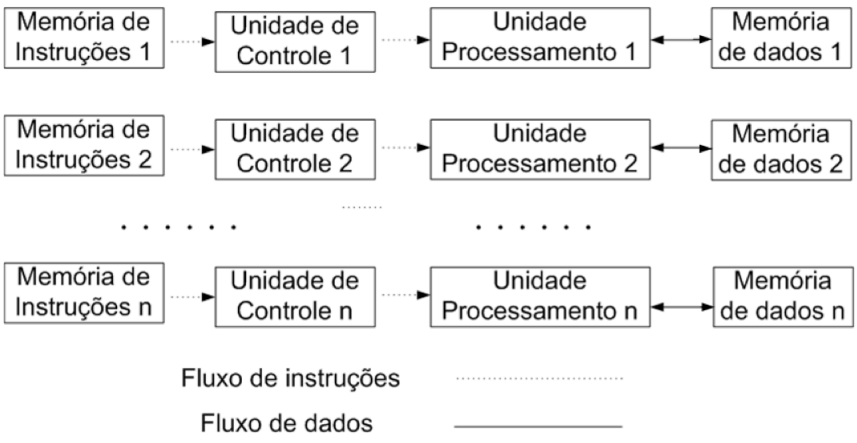


Figura 2.6: Diagrama de blocos de uma arquitetura MIMD.

Na Figura 2.7 pode ser obsevado um exemplo de um conjunto de fluxo de instruções sendo executadas em uma arquitetura MIMD, para cada processador (Px) existe um conjunto distinto de fluxo de instruções a serem executadas.

Este tipo de arquitetura possui a vantagem dos processadores poderem executar várias instruções simultaneamente. Já, em contrapartida, possui a desvantagem do custo de balancea-

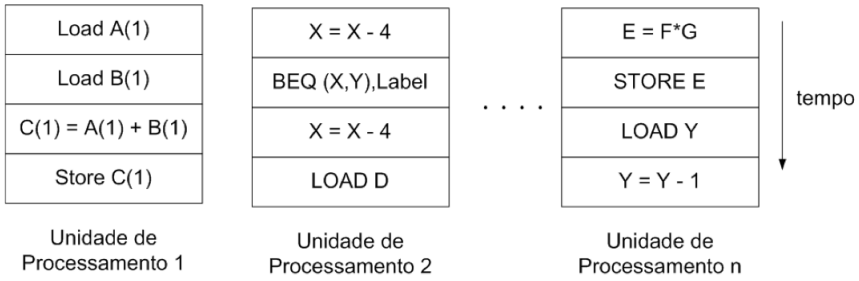


Figura 2.7: Fluxo de instruções sendo executadas na arquitetura MIMD.

mento de carga.

2.2 Conceitos Básicos Sobre Programação Paralela

A programação paralela é uma técnica usada em tarefas grandes e complexas para obter resultados em tempo desejável, dividindo-as em tarefas menores, executadas em vários processadores para serem executadas simultaneamente, como é apresentado na Figura 2.8.

O termo paralelismo em computação é normalmente utilizado para designar paralelismo físico, ou seja, o fato de se ter múltiplos elementos de processamento para executar os processos, sendo que todos eles podem estar ativos simultaneamente (HWANG; XU, 1998).

A concorrência implica em mais de um processo iniciado e não terminado. Não existe uma relação com o número de elementos de processamento utilizados. Obter paralelismo na execução desses processos só é possível quando existe mais de um elemento de processamento, de modo que diversos processos possam estar em execução em um determinado instante.

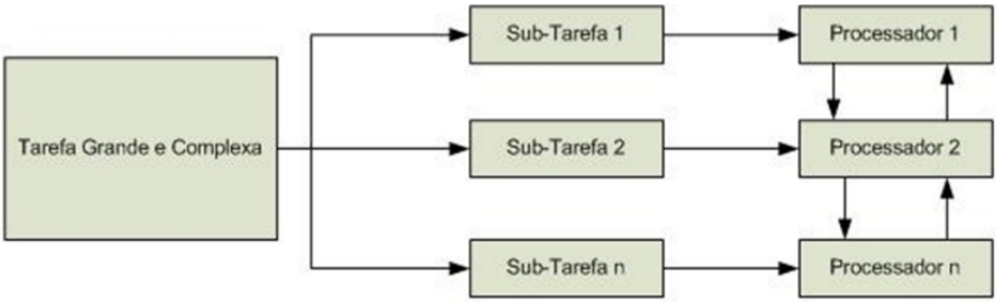


Figura 2.8: Diagrama de Paralelização.

Quando existe apenas um elemento de processamento e vários processos estão sendo executados de maneira concorrente existe um pseudo-paralelismo. O usuário tem a impressão de que os processos estão sendo executados ao mesmo tempo, mas o que realmente acontece é o compartilhamento do elemento de processamento entre os processos em execução. Isto é, em

um determinado instante de tempo, apenas um processo está em execução, enquanto os demais estão aguardando a liberação do processador ou de outro recurso.

Os principais objetivos do paralelismo são:

- Aumentar o desempenho (reduzindo o tempo) no processamento;
- Fazer uso de um sistema distribuído para a resolução de tarefas e
- Obter ganhos de desempenho.

2.2.1 Tipos de Paralelismo Quanto ao Conteúdo

Quanto ao conteúdo a ser paralelizado, pode-se fazer a seguinte classificação: Paralelismo de Dados, Funcional e de Objetos.

2.2.1.1 Paralelismo de Dados

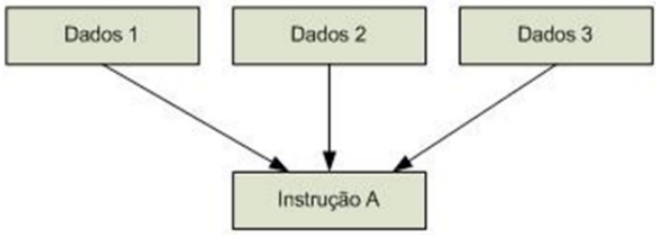


Figura 2.9: Paralelismo de dados.

Neste tipo de paralelismo, o processador executa as mesmas instruções sobre dados diferentes, como mostra a Figura 2.9. Aplicado, por exemplo, em programas que utilizam matrizes imensas e para cálculos de elementos finitos. Exemplos: resolução de sistemas de equações e multiplicação de matrizes.

2.2.1.2 Paralelismo Funcional

Neste caso, o processador executa instruções diferentes que podem ou não operar sobre o mesmo conjunto de dados, como mostra a Figura 2.10. Aplicado em programas dinâmicos e modulares onde cada tarefa é um programa diferente. Exemplos: Paradigma Produto-Consumidor, simulação e rotinas específicas para tratamento de dados (imagens).

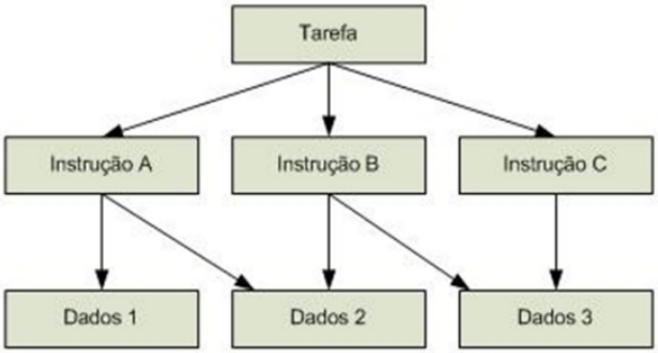


Figura 2.10: Paralelismo functional.

2.2.1.3 Paralelismo de Objetos

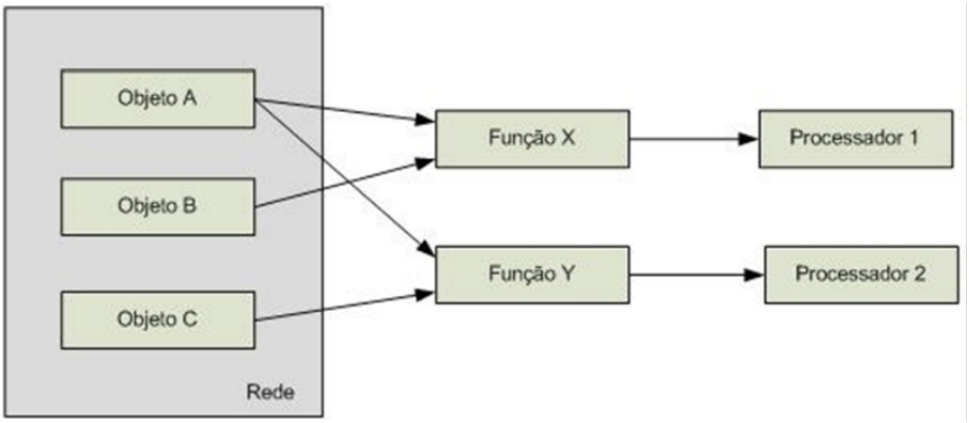


Figura 2.11: Paralelismo de Objetos.

Este é o modelo mais recente, que utiliza o conceito de objetos distribuídos por uma rede (como a Internet), capazes de serem acessados por métodos (funções) em diferentes processadores para uma determinada finalidade, como mostra a Figura 2.11. Exemplo: MPP (*massively parallel processing*) (AMAMIYA et al., 1994; SEONG; AHN; SUNG, 2010; BUSTAMAM; BURRAGE; HAMILTON, 2012).

2.2.2 Tipos de Paralelismo Quanto a Forma

A escolha de um paradigma é fundamental na elaboração de um algoritmo paralelo, para a obtenção do desempenho desejado. Existem 5 tipos de paradigmas (HWANG; XU, 1998):

- Fases Paralelas (Phase Parallel)
- Divisão e Conquista (*divide and conquer*)

- *Pipeline*
- Mestre/Escravo (*Processor Farm*) e
- *Pool* de trabalho (*work pool*)

2.2.2.1 Fases Paralelas

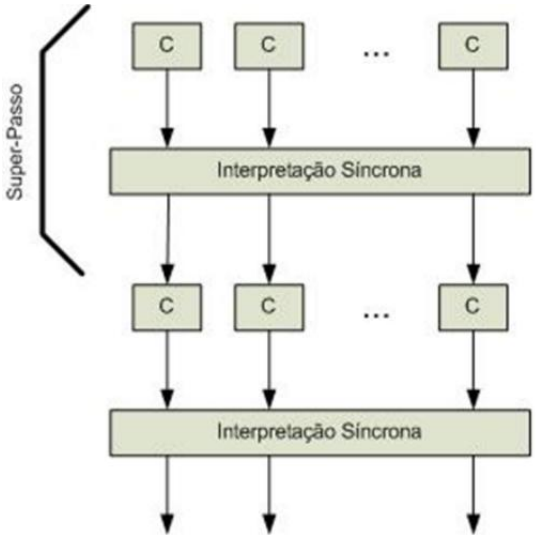


Figura 2.12: Fases Paralelas.

Nesse paradigma, cujo diagrama é apresentado na Figura 2.12, o programa paralelo consiste de um número de super passos e cada super-passo tem duas fases: a fase computacional (C) e a fase de interação, que consiste da sincronização e comunicação. Na fase computacional, múltiplos processos desempenham individualmente uma computação independente C. Na fase de interação subseqüente, os processos desempenham uma ou mais operações de interações síncronas, tais como um barreiras para sincronimos ou uma comunicação de bloco. Então, o próximo super-passo é executado.

Esse tipo de paradigma facilita a detecção de erros e a análise de desempenho, mas tem duas principais falhas: congestionamento nas comunicações e a dificuldade em manter o balanceamento da carga de trabalho entre os processos.

2.2.2.2 Divisão e Conquista

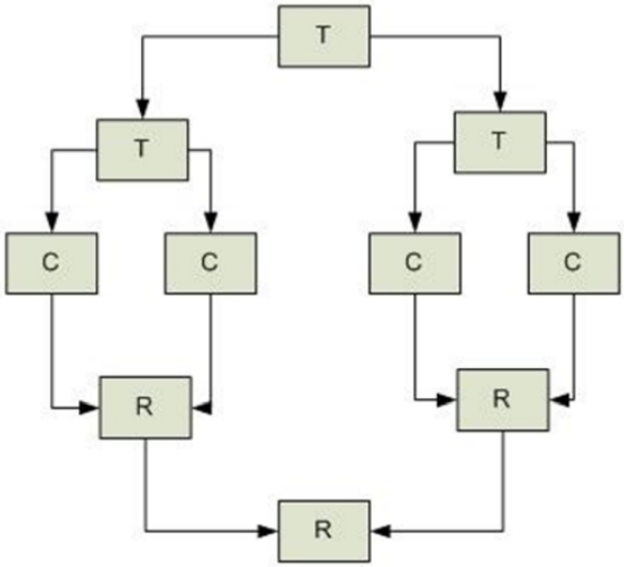


Figura 2.13: Divisão e Conquista.

O paradigma paralelo divisão e conquista *(divide and conquer*), cujo diagrama é mostrado na Figura 2.13, é muito similar a um sistema seqüencial. Um processo pai divide as cargas de trabalho em várias partes pequenas e as atribui para um número de processos filhos.

Os processos filhos, então, computam suas cargas de trabalho em paralelo e os resultados são agrupados pelos pais. A divisão e os procedimentos de agrupamento são feitos recursivamente. Este paradigma é muito natural para computação, mas sua desvantagen é a dificuldade em realizar o balanceamento de carga.

2.2.2.3 *Pipeline*

Neste tipo de paralelismo, a aplicação é constituída de uma seqüência de dados que recebem o processamento em forma de tarefas específicas que devem ser executadas seqüencialmente por processadores distintos, como mostra o diagrama da Figura 2.14.

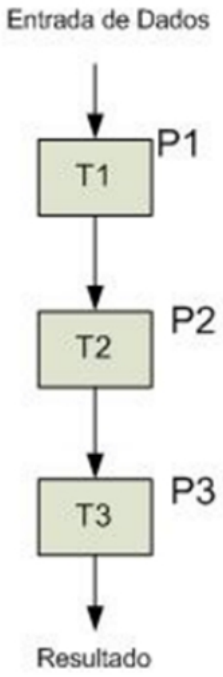


Figura 2.14: *Pipeline*.

Pode-se fazer uma analogia com a construção de uma casa, onde são necessários vários especialistas, como eletricista, engenheiro, encanador, arquiteto, entre outros. Cada um destes tem uma determinada tarefa e alguns têm que esperar o término de outra tarefa para começar a sua, ou seja, cada um é responsável pela sua tarefa e todos trabalham em paralelo, desde que hajam várias casas para serem construídas.

Este paradigma é o mais natural para se desenvolver em paralelo tendo como base um programa sequencial, sendo até possível a sua detecção por compiladores e analisadores de código fonte. Dentre as desvantagens pode-se citar:

- Pouca flexibilidade, visto que modificações no algoritmo podem requerer mudanças drásticas na rede de processadores
- O balanceamento de carga deve ser feito de maneira cuidadosa a fim de que as tarefas específicas lentas não tornem a execução total lenta e
- A relação entre comunicação e execução deve ser baixa a fim de que a sobrecarga de comunicação não torne o processamento ineficiente.

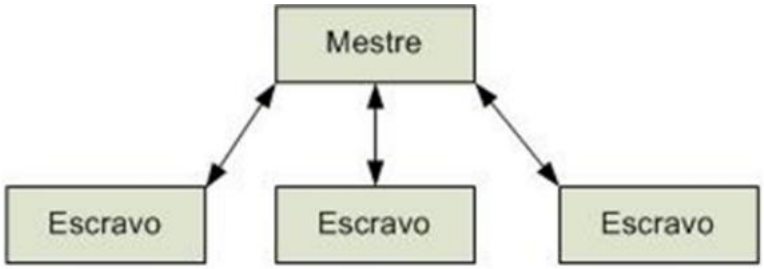


Figura 2.15: Mestre/Escravo.

2.2.2.4 Mestre/Escravo

Neste paradigma, um processador é denominado "mestre" e os demais processadores são denominados "escravos", como é mostrado na 2.15. O processamento consiste de um conjunto de tarefas que são distribuídas pelo processador "mestre" aos processadores "escravos". O processamento "mestre" supervisiona os trabalhos dos "escravos", cada um processando assincronamente as respectivas tarefas.

Este modelo de paralelismo possui várias vantagens como:

- Facilidade de paralelismo do sistema, o que pode ser conseguido por meio do aumento do número de processadores;
- Facilidade de programação e
- Balanceamento de carga mais natural, visto que as tarefas vão sendo submetidas aos processadores de acordo com a disponibilidade.

Como desvantagens, pode-se citar a sobrecarga de comunicação e a possibilidade de gargalo no processador "mestre".

2.2.2.5 *Pool* de Trabalho

Este paradigma é análogo a uma comunidade disposta a executar um trabalho constituído de uma série de tarefas relacionadas numa lista. O paradigma representado pelo diagrama da Figura2.16 é freqüentemente usado no modelo de variável compartilhada. Um conjunto de trabalhos é realizado numa estrutura de dados global.

Inicialmente, pode-se ter somente uma tarefa na lista. Algum processador livre busca essa tarefa e a executa, produzindo zero, uma ou mais novas tarefas que são colocadas na lista. O programa paralelo finaliza quando a lista torna-se vazia.

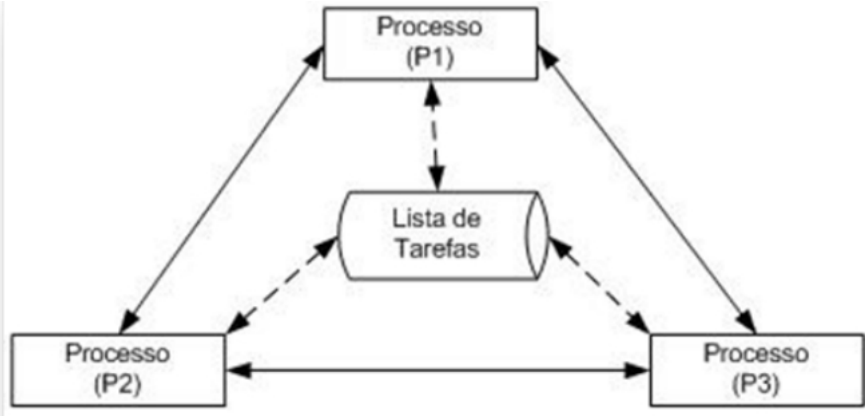


Figura 2.16: *Pool* de trabalho.

Este paradigma facilita no balanceamento de carga, pois a carga de trabalho é dinamicamente alocada para processadores; entretanto, a implementação do modelo de forma a conseguir eficiência, não é fácil, especialmente usando passagem de mensagem.

2.3 Exploração do Paralelismo

Existem alguns problemas a serem analisados para a exploração do paralelismo, pois a eficiência na execução paralela depende do particionamento em módulos dos programas e no escalonamento desses módulos entre os processadores.

Com relação ao problema do particionamento de um programa em tarefas paralelas, procura-se identificar as partições para a obtenção do menor tempo possível na execução, e conseqüentemente, o tamanho adequado para cada uma destas tarefas. Denomina-se o problema do particionamento como a análise da granularidade de tarefas para processamento paralelo.

Pode-se citar duas abordagens para o problema de particionamento de um programa em tarefas a serem executadas paralelamente (MCCREARY; GILL, 1989):

- Paralelismo Explícito: é a detecção do paralelismo pelo programador. Nesta abordagem o programador utiliza linguagens especiais ou bibliotecas para troca de informações entre os processadores. O paralelismo explícito contribui para a diminuição da complexidade dos compiladores para sistemas paralelos pois elimina a necessidade da detecção automática do paralelismo em tempo de compilação.
- Paralelismo Implícito: é a detecção automática do paralelismo. Nesta abordagem, os compiladores devem detectar o paralelismo inerente ao programa desenvolvido nas linguagens seqüenciais convencionais.

2.4 Ambiente Paralelo

O ambiente paralelo pode ser definido como o conjunto de hardware e software para possibilitar o desenvolvimento do processamento paralelo. Um exemplo de ambiente paralelo é a união de vários processadores interligados em rede, como uma plataforma para manipulação de processos paralelos, com um sistema operacional, e uma linguagem de programação, usando um dos modelos de programação paralela.

2.4.1 Granularidade

Granularidade, ou nível de paralelismo, indica a relação entre o tamanho de cada tarefa e o tamanho total do programa, ou seja, é a razão entre computação e comunicação. A granularidade pode ser dividida em três níveis (HWANG; XU, 1998; ALMASI; GOTTLIEB, 1994; KIRNER, 1991; NAVAUX, 1989; HWANG; BRIGGS, 1984):

- Fina: relaciona o paralelismo no nível de instruções, ou operações, e implica grande número de processadores pequenos e simples.
- Grossa: relacionada com o paralelismo no nível dos processos e geralmente se aplica a plataformas com poucos processadores grandes e complexos.
- Média: situa-se em patamar entre as duas anteriores, implicando procedimentos executados em paralelo.

2.5 Construção de Algoritmos Paralelos

Para se construir um algoritmo paralelo, há vários itens que devem ser analisados (CENTURION, 1998), tais como:

- O modelo de paralelismo a ser adotado em função das características da aplicação;
- A forma de distribuição das tarefas nos processadores, para obter um melhor desempenho;
- A ferramenta usada no desenvolvimento do algoritmo paralelo;
- A linguagem a ser usada na implementação do algoritmo e
- Arquitetura na qual se executará o algoritmo, visto que a sua eficiência pode variar de maneira drástica de acordo com o tipo de arquitetura.

Ou seja, para a construção de um algoritmo paralelo é necessário escolher qual a melhor abordagem dentre: começar de um algoritmo seqüencial disponível, já implementado ou a partir das especificações do problema a ser resolvido com a computação paralela.

Ao definir o algoritmo paralelo, deve-se identificar os processos que serão executados em paralelo e verificar se a divisão é conveniente, considerando as comunicações entre processos e tamanho dos processos necessários. Após ter definido os processos, deve-se organizar estes em forma de programas para isto, deve-se conhecer a arquitetura, o algoritmo proposto e o modelo de paralelismo.

2.5.1 Implementação de Programas Paralelos

Para se expressar o paralelismo durante a implementação do programa deve-se utilizar as formas para ativação de processos paralelos, como a sincronização entre os processos, bem como a comunicação.

2.5.1.1 Mapeamento

Nessa etapa deve-se fazer a alocação dos processos nos processadores e definir como será a comunicação por meio de:

- Escalonamento: forma como os processos são colocados e retirados dos processadores.
- Balanceamento de Carga: fator considerado para que um processador não seja sobrecarregado enquanto outro está ocioso.
- Migração de Processos: considera a troca de uma tarefa em execução para outro processador em tempo de execução.

2.6 Avaliação do Desempenho

Ao passar por todas as etapas acima, deve-se medir o desempenho do programa. Este desempenho permite analisar o ganho obtido com o aumento do total de processadores utilizados. Algumas medidas são usadas, como a eficiência e o *speed-up* (ganho de desempenho).

Uma característica fundamental da computação paralela é o aumento de velocidade de processamento por meio da utilização do paralelismo. Para se medir o ganho com o aumento

de processadores pode-se utilizar algumas medidas como speed-up e eficiência(QUINN, 1987; SOUZA, 1996).

2.6.1 *Speed-up*

Speed-up é uma medida utilizada para determinar o aumento de velocidade obtido para a execução de um programa utilizando p processadores, em relação a sua execução seqüencial, usando apenas um processador, sendo obtido pela Equação 2.1.

$$S_p = \frac{T_{seq}}{T_{par}} \tag{2.1}$$

O caso ideal é quando o $S_p = p$, isto é, a velocidade de processamento aumenta linear e proporcionalmente à quantidade de processadores utilizados. Mas existem fatores que degradam essa situação ideal:

- A sobrecarga de comunicação
- Algoritmos parcialmente paralelizáveis
- A dificuldade de balanceamento de carga entre os processadores e
- Casos onde a granularidade é inadequada para o tipo de arquitetura utilizada

Existem também casos em que o *speed-up* alcançado é maior que o ideal, isto é, $S_p > p$. Esse fenômeno é chamado de anomalia de *speed-up* ou *speed-up* super linear.

2.6.2 Eficiência

Eficiência é uma medida que mostra o quanto o paralelismo foi explorado no algoritmo e é dado pela Equação 2.2

$$E_p = \frac{S_p}{p} \tag{2.2}$$

A eficiência quantifica a utilização do processador. Variando entre 0 e 1, o caso ideal é verificado quando $E_p = 1$, indicando uma eficiência de 100%.

2.7 Ferramentas para Programação Paralela

As ferramentas para programação paralela devem ser eficientes e de fácil utilização para comunicação e sincronização entre os processos e a ativação de processos paralelos. A escolha do tipo de ferramenta que será utilizada para o desenvolvimento de um programa paralelo é um decisão extremamente importante, devendo levar em consideração vários fatores; entre os quais: a aplicação, o usuário que utilizará a ferramenta e a arquitetura que executará os códigos gerados.

Pode-se citar alguns tipos de ferramentas usadas na construção de programas paralelos: ambientes de paralelização automática, extensões paralelas para linguagens seriais e linguagens concorrentes.

2.7.1 Compiladores Paralelizantes

Compiladores paralelizantes caracterizam ambientes de paralelização automática, sendo responsáveis pela geração automática de programas paralelos a partir de sua versão seqüencial. A paralelização automática é normalmente a possibilidade mais simples de ser utilizada, uma vez que requer pouco ou até nenhum esforço extra do usuário, não sendo preciso nem modificar o programa, nem aprender uma nova linguagem. Entretanto, o desempenho obtido geralmente é modesto, a não ser em aplicações específicas onde o *speed-up* pode ser grande. Além disso, nem sempre esses compiladores estão disponíveis, principalmente para sistemas de memória distribuída. Normalmente exploram granularidade fina e apresentam baixa flexibilidade.

2.7.2 Extensões Paralelas

Extensões paralelas são bibliotecas que contêm um conjunto de instruções que complementam linguagens seqüenciais já existentes. Estes ambientes requerem mais trabalho do programador, mas ainda evitam a necessidade de aprendizagem de uma nova linguagem e a completa reescrita do código fonte (o usuário deve aprender uma extensão para uma linguagem já conhecida). Geralmente, o desempenho obtido é superior ao obtido pelos compiladores paralelizantes. Alguns dos exemplos de extensões paralelas são P4 (CARRIERO; GELERNTER, 1989), PVM (SUNDERAM et al., 1994), MS-MPI entre outros.

É importante salientar que normalmente esses ambientes utilizam a passagem de mensagens como mecanismo de comunicação, visto que formam uma arquitetura MIMD com memória distribuída. Entretanto, é possível que seja simulada uma memória compartilhada, ou ainda que

o ambiente utilize memória compartilhada quando necessário de modo transparente à aplicação paralela.

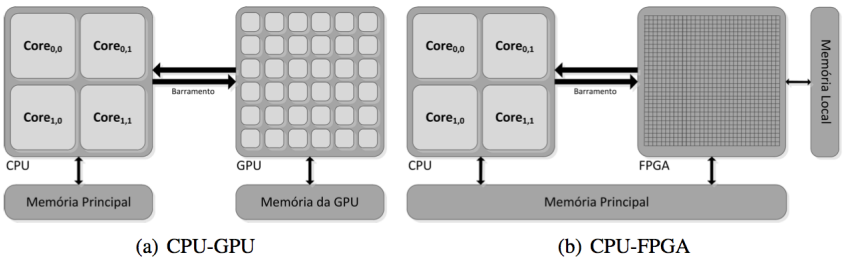
2.7.3 Linguagens Concorrentes

O terceiro tipo de ferramenta está relacionado às linguagens criadas especialmente para o processamento paralelo, o que implica em tempo de aprendizagem de uma linguagem totalmente nova e restrita total do código fonte. Essas ferramentas tendem a fornecer melhor desempenho, de maneira a compensar a sobrecarga sobre o programador.

Outra vantagem destas linguagens é que geralmente elas possibilitam a construção de código bem estruturado, tornando fácil a identificação dos processos que estão executando em paralelo e a comunicação entre eles. Exemplos dessas linguagens são: C Paralelo, HPF e CPAR.

2.8 Arquiteturas Paralelas Híbridas

De forma a atender uma demanda crescente por poder computacional, os arranjos computacionais tradicionais, formados por aglomerados de processadores homegêneos, estão passando por uma transformação, utilizando uma configuração híbrida formada por processadores *multicore* e de coprocessadores especializados, como GPUs (Figura 2.17(a)) ou FPGAs (Figura 2.17(b)) (ASANOVIC et al., 2009), o que faz com que o ambiente de desenvolvimento seja formado por ferramentas customizadas para cada tecnologia utilizada (PINTO, 2011). Na Figura 2.17 são apresentados dois exemplos de arquiteturas paralelas híbridas com algumas de suas particularidades, onde a Figura 2.17(a) está sendo utilizado um GPU como coprocessador paralelo, e na Figura 2.17(b) um FPGA.



(a) CPU-GPU (b) CPU-FPGA

Figura 2.17: Exemplos de arquiteturas híbridas (PINTO, 2011).

Um processador *multicore* é uma unidade de processamento de uso geral composta por dois ou mais núcleos completos de processamento em um mesmo chip, onde cada um destes núcleos

é interpretado como um um processador lógico independente pelo Sistema Operacional. Desta forma é possível que sejam executadas diferentes aplicações em cada um dos núcleos, ou que uma aplicação faça a distribuição de sua carga de processamento entre os núcleos, buscando uma execução mais eficiente. Podem ser utilizadas como ferramentas de desenvolvimento:

- OpenMP (*Open Multi-Processing*) (CHAPMAN; JOST; PAS, 2007): que utiliza um conjunto de diretivas de compilação para identificar trechos do código (C, C++ e Fortran) passíveis de paralelização, como loops ou seções;
- Cilk (CILK..., 2009; BLUMOFE, 1996): uma extensão da linguagem C que utiliza políticas de escalonamento chamadas de *work stealing*, por meio de duas palavras chave *spawn*, para indicar o procedimento que será executado em paralelo, e *sync* para criar uma barreira de espera para a conclusão dos procedimentos que estejam sendo executados em paralelo; e a
- Intel TBB (Threading Building Blocks) (REINDERS, 2007): que é uma biblioteca de *templates* para a programação paralela em C++.

Para auxiliar o processador *multicore* podem ser utilizados dispositivos especializados no processamento paralelo que funcionam como coprocessadores, tais como GPUs e FPGAs. Detalhes da arquitetura do GPU e do desenvolvimento de aplicações utilizando uma ferramenta de programação de um ambiente CPU-GPU, denominado CUDA (*Compute Unified Device Architecture*), podem ser encontrados nas Seções 2.12 e 2.12.2, respectivamente.

Um FPGA (*Field Programmable Gate Arrays*) é um dispositivo de *hardware* reconfigurável que pode ser reprogramado para resolver quaisquer tipos de processamentos (SHAN, 2006; PINTO, 2011), sendo formado por três tipos de elementos: blocos lógicos programáveis e de memória, roteamento e blocos de entrada e saída (GOKHALE; GRAHAM, 2005). O desenvolvimento das aplicações de um FPGA se dá, normalmente, por meio de linguagens de descrição de *hardware* conhecidas como HDL (*Hardware Description Languages*), sendo as duas comumente utilizadas a VHDL (ACCELLERA, 2006) e a Verilog (VERILOG, 2012), que permitem a descrição e o funcionamento de blocos de circuitos lógicos.

Como pode ser observado, o desenvolvimento de aplicações para arquiteturas paralelas híbridas não é uma atividade trivial, devendo ser utilizadas diferentes ferramentas, uma para cada tecnologia, além da implementação de pontos para a transferência de dados entre os ambientes de execução dos dispositivos. Para facilitar esta atividade foram criados ambientes de execução com suporte a arquiteturas paralelas híbridas, buscando o aproveitamento eficiente dos recursos. Entre estes ambientes podem-se destacar: XKaapi (*Kernel for Adaptative, Asynchronous*

Parallel and Interactive programming): uma biblioteca em C++ que permite a execução de múltiplas threads com fluxo de dados sincronizado, tendo sido implementado essencialmente para as arquiteturas NUMA (Non-Uniform Memory Access) e UMA (Unified Memory Access) e possuindo inspiração no Cilk(MENTEC; GAUTIER; DANJEAN, 2011; LIMA et al., 2012), e o StarSs que é baseado na adição de diretivas de paralelização do código sequencial C e Fortran (LABARTA, 2010; AYGUADE et al., 2009).

O objeto comum das bibliotecas/ambientes de execução para arquiteturas paralelas híbridas é a busca pela otimização da execução da aplicação por meio de algoritmos de escalonamento e balanceamento de carga além do gerenciamento das alocações e transferência de dados entre as memórias dos recursos, como apresentado na Figura 2.18.

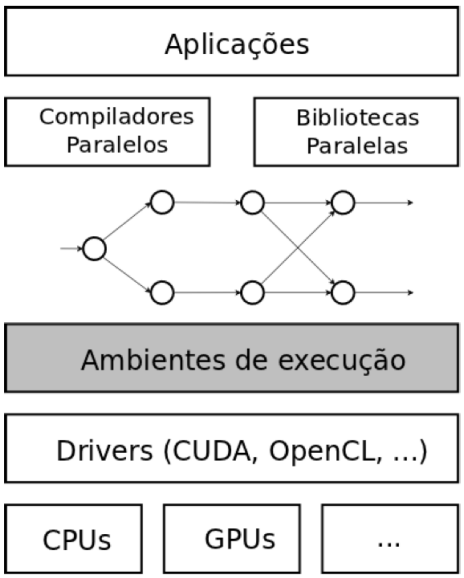


Figura 2.18: Ambiente de execução de uma arquitetura paralela híbrida (PINTO, 2011).

Nas seções seguintes são apresentadas as pesquisas bibliográficas relativas à plataforma CoP-WS, apresentando o XML (EXtensible Markup Language) que é uma linguagem de marcação para a transferência de dados, que é utilizada para a configuração dos protocolos usados pela tecnologia de interoperabilidade WS, quanto para a transferência de dados entre clientes e servidores. Em seguida é descrita a tecnologia WS que é utilizada para a criação de uma interface de interoperabilidade entre os nós computacionais do arranjo paralelo híbrido. Posteriormente, é descrito o modelo de programação MapReduce utilizado na distribuição dos dados entre os nós computacionais responsáveis pela mesma funcionalidade, em diferentes estágios de fluxo de processamento da aplicação. Para o processamento usando a plataforma CoP-WS o conceito de MapReduce é utilizado na distribuição dos dados entre os nós. As seções restantes referem-se à descrição do GPU, usado como coprocessador na plataforma proposta, seguida da

descrição sobre a área de Reconhecimento de Padrões e da teoria de Near Sets, acompanhada de GLCM (Gray Level Co-occurrence Matrix), utilizada na aplicação de reconhecimento de explosões solares. Finaliza-se o capítulo com a descrição de fundamentos sobre a radiointerferometria e o BDA (Brazilian Decimetric Array), um arranjo interferométrico em construção pelo INPE (Instituto Nacional de Pesquisas Espaciais), que inspirou as aplicações de viabilidade da presente tese.

2.9 XML - *EXtensible Markup Language*

XML (*EXtensible Markup Language*) é uma linguagem de marcação que têm como objetivo criar um estrutura para transporte e armazenamento de dados, sendo um dos padrões mantidos pelo W3C (*World Wide Web Consortium*), organização responsável pela manutenção dos padrões abertos utilizados na Web. A principal diferença entre o XML e o HTML (*HyperText Markup Language*) (W3C, 2013) esta no objetivo de cada uma destas linguagens, sendo suas *tags* especificadas em tempo de projeto, ou seja, é o desenvolvedor que é responsável pelo significado/conteúdo a ser utilizado por cada *tag* criada. O XML possui como foco a representação do dado, de forma que este possa ser transportado e/ou armazenado, já o HTML têm seu foco na apresentação, ou seja, como os dados/informações serão apresentados. Na Listagem 2.1 é apresentado um pequeno exemplo do código XML.

Algoritmo 2.1 Exemplo de código em XML (W3C, 2007b).

```
1 <note>
2 <to>Tove</to>
3 <from>Jani</from>
4 <heading>Reminder</heading>
5 <body>Don't␣forget␣me␣this␣weekend!</body>
6 </note>
```

Como pode ser observado no Algoritmo 2.1, o XML utiliza um conjunto de etiquetas, conhecidas como *tags*, para indicar blocos representativos da estrutura da informação, que no exemplo apresentado são: *note*, *to*, *from*, *heading*, e *body*. Estas *tags* encontram-se representados pelos símbolos < e >, e tem como final de sua marcação os símbolos </ e >, de forma que o exemplo seja a representação de uma nota (*note*), para *Tove* (*to*) de *Jani* (*from*) que tem como cabeçalho (*heading*) *Reminder*, e o corpo do texto (*body*) *Don't forget me this weekend!*.

Desta forma o XML permite que os dados sejam armazenados juntos com sua estrutura, como um "esqueleto" do dado. Por um lado isso facilita a comunicação entre sistemas, quando comparado com arquivos texto onde cada linha é um registro de dados, e cada posição da linha

(coluna inicial+final) indicam um campo/dado, que *a priori* necessita se ter um conhecimento prévio. Por outro lado, o arquivo XML ainda é um formato texto, o que faz com que não exista, na sua geração uma regra rígida que faça com que os dados sigam uma formatação específica, o que pode causar problemas quando os dados são gerados em uma estação e é enviada para outra.

Contudo o XML possui um recurso auxiliar o que fornece a capacidade de validação da definição da estrutura adotada para o arquivo XML, para isso são utilizados arquivos de definição de documentos, DTD (*Document Type Definition*). No Algoritmo 2.2 é apresentada a DTD do arquivo XML apresentado no Algoritmo 2.1. A utilização da DTD permite que seja definido um metadado, ou seja, uma representação abstrata dos dados que serão representados pelo arquivo XML.

Algoritmo 2.2 Exemplo de um arquivo DTD (W3C, 2007a).

```
1 <!DOCTYPE note [
2 <!ELEMENT note (to,from,heading,body)>
3 <!ELEMENT to (#PCDATA)>
4 <!ELEMENT from (#PCDATA)>
5 <!ELEMENT heading (#PCDATA)>
6 <!ELEMENT body (#PCDATA)>
7 ]>
```

Como pode ser observado no Algoritmo 2.2, a DTD contem a especificação do registro, ou da estrutura da informação, a ser armazenada/transferida. A tag *!DOCTYPE note* define o elemento principal da informação, que neste exemplo é a notificação (*note*), que por meio da *!ELEMENT note* é definida como sendo composta pelos elementos *to, from, heading* e *body*. A partir deste ponto a DTD utiliza o tag *!ELEMENT* para definir os campos e o tipo de do dado a ser armazenado neste campo, por exemplo *#PCDATA*, que é uma string de caracteres.

Com a utilização da DTD, as linguagens que possuem recursos para a manipulação de arquivos no formato XML, podem fazer um confronto entre os dados existentes no arquivo que seguem a formatação indicada pelo DTD, antes do início do processamento dos dados. Isso aumenta a confiabilidade da qualidade dos dados trafegados/representados utilizando este tipo de tecnologia.

Trabalhos como o de Gonzalez (GONZALEZ, 2012), apresenta o XML como sendo uma ferramenta de transporte de dados entre diferentes plataformas de forma a se obter um mecanismo de trabalho colaborativo entre diferentes sistemas, uma vez que é possível fazer a definição do metadados e criar a especificação da informação a ser utilizada entre os sistemas participantes. Essa visão de metadados em ambiente distribuído também pode ser visto em trabalhos como do

Zhang e de seus colaboradores (ZHANG; LANG; DUAN, 2011), que trabalharam com o armazenamento de grandes volumes de dados distribuídos, organizados em documentos estruturados em XML e implementaram uma ferramenta de busca usando o modelo de programação MapReduce (que será discutido na Seção 2.11).

2.10 WS - *Web Services*

Exemplos de utilização de WS pela comunidade científica podem ser encontrados desde a década de 1995. Contudo, sua utilização tem sido vista como sendo uma ferramenta de acesso a serviços oferecidos por um determinado grupo de pesquisa, ou seja, existe um determinado serviço instalado em um servidor, ou mesmo em um cluster, como é o caso do MolDynGrid (SALNIKOV et al., 2009), mas o serviço oferecido é especializado e completo, o que configura uma comunicação entre uma estação cliente e o servidor onde WS que oferece o serviço de interesse.

Um WS (Web Service) é um sistema computacional projetado para suportar a interoperabilidade, na interação entre duas máquinas, ponto a ponto, em uma rede. Para a descrição do serviço oferecido por uma entidade provedora, aqui representada por um servidor, é utilizada uma interface que descreve as características deste, por meio do WSDL (*Web Service Description Language*). O sistema que interage com o WS, ou seja, a entidade requisitante, utiliza a troca de mensagens SOAP (*Simple Object Access Protocol*), transmitida usando HTTP (*Hyper-Text Transfer Protocol*) com a serialização do XML e outros padrões relacionados com a Web (LEE, 2008; PAPAZOGLOU, 2008; THOMAS; THOMAS; GHINEA, 2003; BOOTH et al., 2004).

Desta forma, um WS pode ser visto como sendo a abstração dos conceitos que devem existir em uma implementação real de um agente e de um serviço, onde por agente deve-se entender uma peça de um software ou hardware capaz de enviar e receber mensagens, enquanto que um serviço é a abstração de um recurso que represente a capacidade de processamento de uma tarefa, dentro de um contexto onde existem as entidades provedora e requisitante, que são responsáveis pela oferta e consumo deste serviço respectivamente (BOOTH et al., 2004).

A tecnologia base de um WS é a utilização do XML e do protocolo de comunicação HTTP. O XML é utilizado pelo WS por dois dos protocolos básicos desta plataforma, o WSDL (*Web Services Description Language*) (W3C, 2001) e pelo SOAP (*Simple Object Access Protocol*) (W3C, 2007c) que são responsáveis pela descrição do funcionamento do serviço publicado pelo servidor, e pela troca de mensagens durante o consumo de um serviço, respectivamente. Um fluxo de comunicação entre um cliente, consumindo um serviço, e um servidor, oferecendo um

serviço, pode ser visto na Figura 2.19.

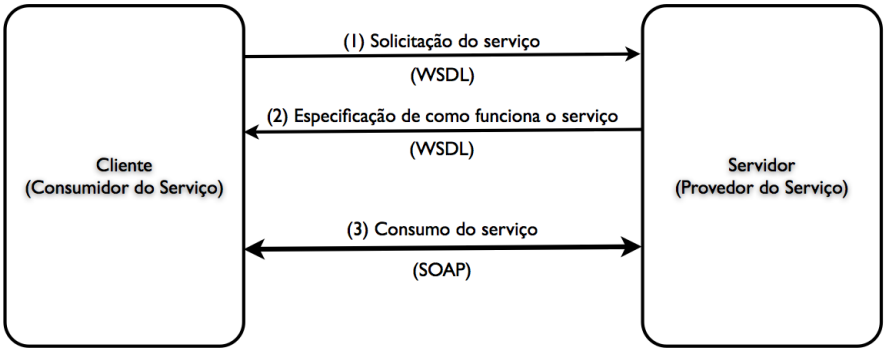


Figura 2.19: Fluxo de comunicação entre um Cliente e um servidor Web Service.

Na Figura 2.19 pode ser visto que antes do consumo de um serviço, por parte de um cliente, existe uma etapa de conhecimento do funcionamento deste serviço, representado pelas chamadas 1 e 2 e que utilizam o protocolo WSDL, em seguida tem início o consumo do serviço por meio do protocolo SOAP, tanto na chamada do serviço, no sentido do cliente para o servidor, quanto no retorno do processamento, no sentido do servidor para o cliente.

Como pode ser observado na Figura 2.19, em um ambiente operacional de um WS existem dois atores, que são o cliente (entidade requisitante), que consome o serviço oferecido e um servidor (entidade provedora). Contudo este cenário pode ser mais sofisticado com a adição de um terceiro ator, conhecido como *Service Broker*. Este ator funciona como um roteador de serviços, e é utilizado quando um cliente não sabe o endereço dos servidores que oferecem um determinado serviço (Figura 2.20). Os servidores por sua vez devem fazer o registro de seus serviços neste *Broker*, para que possam ser localizados por clientes interessados em seus serviços (THOMAS; THOMAS; GHINEA, 2003).

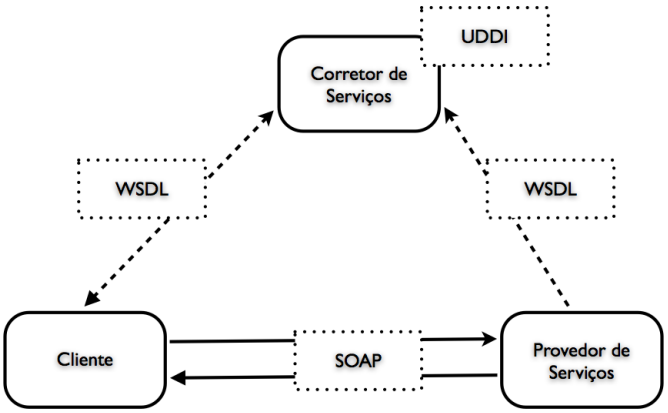


Figura 2.20: Fluxo de comunicação de um WS: Cliente, Servidor e Broker.

Como observado na Figura 2.20, o protocolo de comunicação, tanto do cliente quanto do WS com o *Broker* é o WSDL, o UDDI (*Universal Description Discovery and Integration)* fun-

ciona como um banco de dados especializado no registro e na publicação de serviços oferecidos por servidores dentro de uma rede.

2.10.1 WSDL - *Web Service Description Language*

WSDL é um documento escrito em XML que tem por objetivo especificar um serviço, ou uma lista de serviços, oferecido por um WS. Neste documento são encontradas informações como:

- Descrição do serviço;
- Especificação de como acessar este serviço; e
- Quais são as operações e/ou métodos disponíveis.

Em um documento WSDL os serviços são definidos como sendo uma coleção de endereços de rede (*endpoints*), onde para cada endereço de rede é definido um port com um *binding* reutilizável, e a mensagem como dados a serem trocados durante a comunicação. A Tabela 2.1 apresenta uma relação dos elementos que fazem parte do documento WSDL, ou seja, as *tags* XML que descrevem o serviço oferecido pelo WS: <type>, <message>, <portType> e <binding>.

Elementos	Descrição
<type>	Um listagem dos tipos de dados utilizados pelo WS
<message>	Definição dos tipos de dados comunicados, elementos de uma operação
<portType>	Conjunto de operações que serão suportadas pelos *endpoints*
<binding>	Especificação do protocolo e formato do dado para um particular tipo de port.

Tabela 2.1: Estrutura dos elementos de um documento WSDL (W3C, 2008b).

Na Listagem 2.3 é apresentada uma parte do documento WSDL, e neste código pode ser vista a descrição de uma operação (serviço) chamada *getTem* (linhas 9 a 14) que possui como parâmetro de entrada uma *string* de nome *term* (linhas 1 a 3), que apresenta como reposta uma outra *string* com o nome de *value* (linhas 5 a 7).

2.10.2 SOAP - *Simple Object Access Protocol*

SOAP é o protocolo utilizado para realizar a troca de dados, de forma estruturada, em uma plataforma distribuída, que é utilizada pelo WS, assim como o WSDL, o SOAP também é um documento XML, formado pelos seguintes elementos (W3C, 2008a):

Algoritmo 2.3 Exemplo de código de um documento WSDL (W3C, 2008b).

```
<message name="getTermRequest">
        <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
        <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
        <operation name="getTerm">
                <input message="getTermRequest"/>
                <output message="getTermResponse"/>
        </operation>
</portType>
```

- Envelope (envelope): é a estrutura do documento XML com o SOAP;
- Cabeçalho (header): contem as informações do cabeçalho da mensagem;
- Corpo (body): contém as informações das chamadas e das respostas realizadas; e
- Erro (Fault): contem as mensagens de erros e as informações de status.

Na Listagem 2.4 é apresentada a estrutura do documento XML de um pacote SOAP com os elementos descritos.

Algoritmo 2.4 Estrutura de um documento SOAP (W3C, 2008a).

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
        ...
</soap:Header>

<soap:Body>
        ...
        <soap:Fault>
                ...
        </soap:Fault>
</soap:Body>

</soap:Envelope>
```

2.11 Modelo de programação MapReduce

MapReduce é um modelo de programação que tem seu foco no processamento de grandes volumes de dados em ambientes distribuídos. Sua inspiração vem da função de mesmo nome da linguagem funcional Lisp, e tem sua lógica de processamento baseada em dois estágios de processamento: 1) Map; e 2) Reduce. Na Figura 2.21 é apresentado um exemplo da lógica de funcionamento de uma aplicação implementada neste modelo, onde o objetivo da aplicação é realizar a contagem do número de ocorrências de uma determinada palavra dentro da massa de dados de entrada..

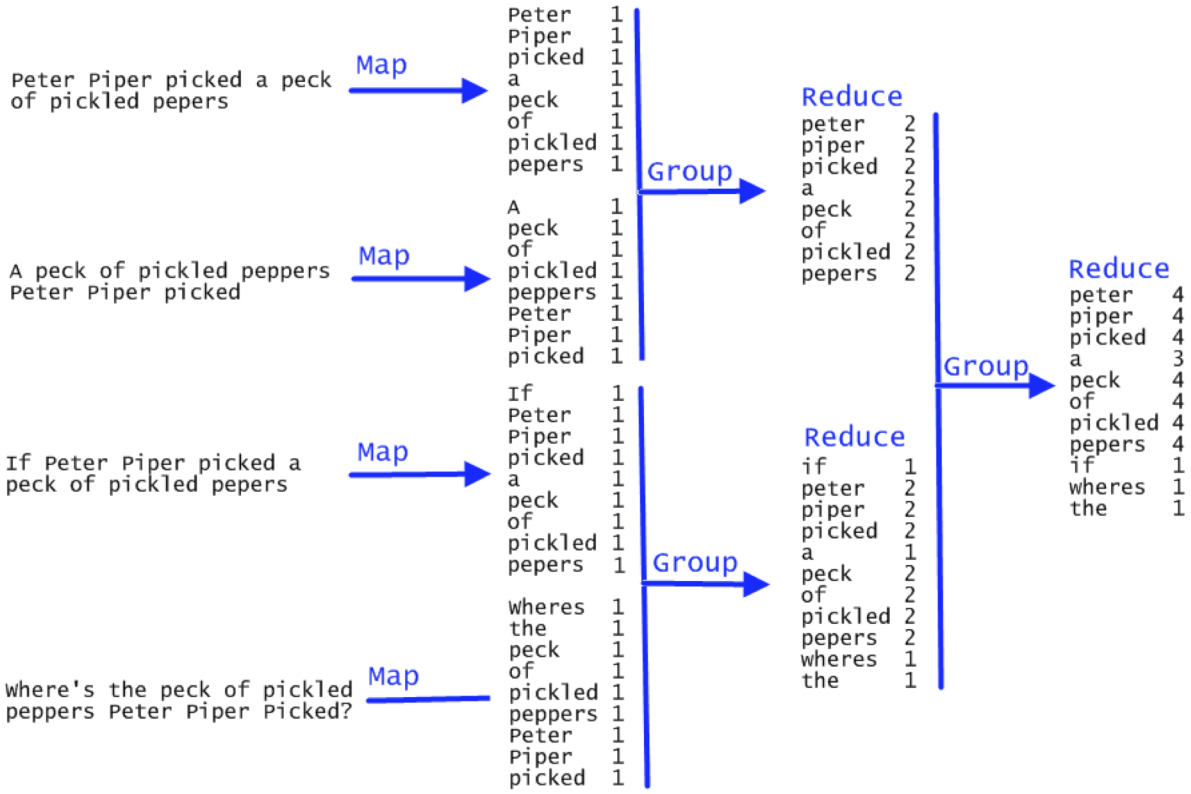


Figura 2.21: Exemplo de uma operação de MapReduce (BARFORD, 2010).

Como pode ser observado na Figura 2.21 o primeiro estágio de processamento, mostrado na metade esqueda, é a execução da função Map, que tem por objetivo organizar os dados em um par *Chave/Valor*, onde, neste exemplo a chave utilizada foi cada uma das palavras da massa de dados de entrada e o valor é o número de ocorrências desta palavra (chave).

A partir deste ponto a estrutura de dados formada pela função Map, passa por uma série de estágios de processamento da função Reduce, mostrado na metade direita da Figura 2.21, que no exemplo apresentado foram necessárias duas chamadas, onde cada execução da função reduce usa como entrada a saída da função anterior. O final do processamento é um conjunto de valores

chave/valor com o resultado do processamento. Um exemplo possível para a implementação da função Map desta aplicação pode ser vista na Listagem 2.5, e para a função Reduce na Listagem 2.6.

Algoritmo 2.5 Código exemplo da função Map.

```
1 map(String key, String value):
2         for each word w in value:
3                 EmitIntermediate(w,"1");
```

Algoritmo 2.6 Código exemplo da função Reduce.

```
1 reduce(String key, Interator values):
2         int result = 0;
3         for each v in values:
4                 result += ParseInt(v);
5                 Emit(AsStrig(result));
```

Por se tratar de um modelo de programação, pode ser utilizada qualquer linguagem que porte este tipo de implementação, ou seja, a utilização do modelo de programação está no projeto e implementação da aplicação. Contudo existem *frameworks* especializados neste tipo de processamento, como é o caso do Hadoop (APACHE, 2013c) mantido pela Apache Software Fundation(APACHE, 2013a), que permite a utilização de um ambiente distribuído, por meio de um sistema de arquivo especificamente implementado para esta finalidade o HDFS (*Hadoop Distributed File System*) (APACHE, 2013b), além de permitir a utilização de uma série de linguagens de programação, como Java, C/C++, Python e Ruby por exemplo. Uma análise do sistema HDFS e de aplicações utilizando MapReduce pode ser encontrada no trabalho de Kaushik e seus colaboradores (KAUSHIK; BHANDARKAR; NAHRSTEDT, 2010).

Na Tabela 2.2 é mostrada a evolução do uso de aplicações implementadas com o modelo MapReduce pela Google durante os anos de 2004 a 2009, em termos de número de tarefas, média de tempo por tarefa, número de servidores usados, volume de dados de entrada, processados e saída e média de tarefas processadas por servidor.

2.12 GPGPU

Pressionado pela indústria de Jogos nos últimos anos, o GPU (*Graphic Processing Unit*), passou por uma transformação, deixando de atuar como uma simples placa gráfica, para assumir o papel de coprocessador apoiando o CPU em atividades ligadas ao processo de renderização de vídeo, entre outras atividades gráficas, e até mesmo de ações físicas realizadas por personagens e elementos existentes no jogos.

	Ago-04	Mar-06	Set-07	Set-09
Número de tarefas	29.000	171.000	2.217.000	3.467.000
Média tempo (seg)	634	874	395	475
Número servidores usados	217	2.002	11.081	25.562
Dados de entrada lidos (Tera)	3.288	52.254	403.152	544.130
Dados em processamento (Tera)	758	6.743	34.774	90.120
Dados de saída escritos (Tera)	193	2.970	14.018	57.520
Média de número de servidores por job	157	268	394	488

Tabela 2.2: Evolução do uso de aplicações implementadas usando o modelo de programação MapReduce pela Google (BARROSO; HÖLZLE, 2009).

A partir do ano de 2001, com o modelo do GPU GeForce 3 da empresa NVIDIA, passou a ser possível a interceptação do *pipeline* do processamento gráfico, e ter acesso aos processadores internos do GPU, de forma que este processadores pudessem realizar processamentos "genéricos", previamente definidos por um programador, ou seja, não relacionados com o processamento gráfico. Esta mudança de escopo dentro do contexto do GPU fez com que sua arquitetura chamasse a atenção da comunidade científica que passou a olhar para o GPU não mais como um dispositivo gráfico, mas sim como um dispositivo de processamento massivamente paralelo, capaz de auxiliar a comunidade em processamento de grandes volumes de dados (WU; LIU, 2008). Assim surgiu a sigla GPGPU com o significado de *General-Purpose on GPU*.

2.12.1 Evolução do GPU como Dispositivo de Programação Paralela

Entre os anos de 2001 e 2005, era possível utilizar o GPU, como um dispositivo de computação genérica, ou seja, computação não gráfica. Mas era necessário que fossem realizadas intervenções no pipeline gráfico da placa.

Na Figura 2.22 são apresentadas as duas posições nas quais eram possíveis inserir código para o processamento dentro do GPU. Os dois grupos de processadores programáveis no GPU eram de: *Vertex*, responsável pela determinação dos vértices de uma imagem, e *Fragment*, que realiza sintese da imagem (*rendering*).

A ideia consistia em passar aos *Vertex* quatro pontos que indicavam as dimensões da matriz cujos elementos deveriam ser calculados. A matriz gerada pelo *Vertex* tinha seus valores processados pelos *Fragment*.

Desde 2003, a indústria de semicondutores tem desenvolvido dois modelos de processadores, de forma a continuar a evolução do desempenho dos dispositivos computacionais (HWU; KEUTZER; MATTSON, 2008); os *multicores* e os *many-cores*. Os *multicores,* cujos processado-

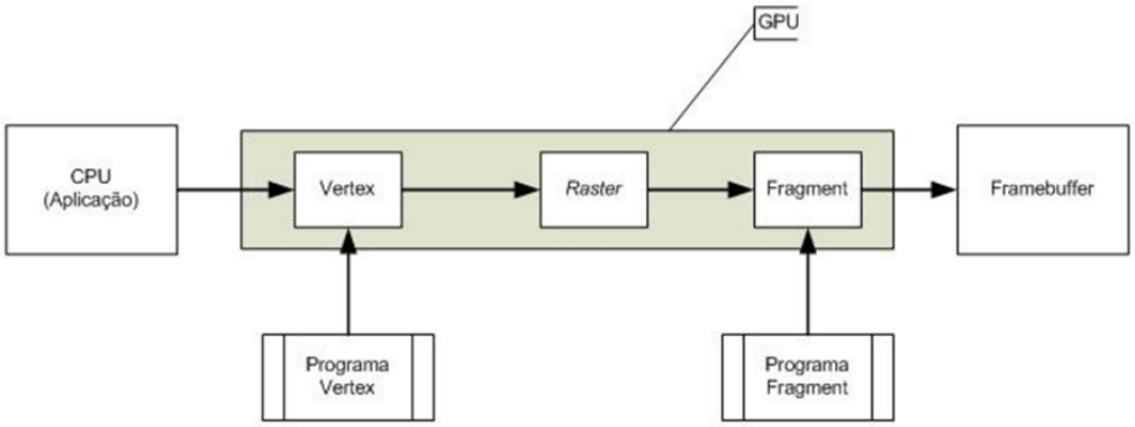


Figura 2.22: Pipeline do desenvolvimento tradicional em GPU.

res são usados pelos CPUs, procuram aumentar a velocidade de processamento das aplicações sequenciais por meio da paralelização de instruções obtida com a distribuição da carga de processamento entre os núcleos existentes.

Estes começaram com dois núcleos (dual core) e possuem como recente representante o processador da Intel i7, que possui quatro núcleos de processamento, cada um dos quais com a capacidade de executar múltiplas instruções fora de ordem, com uma implementação completa de instruções x86, também suportando *hyperthreading* com dois segmentos de *hardware*.

Já a arquitetura de *many-core,* cujos processadores são usados pelos GPUs, tem como foco a otimização da taxa de transferência de dados para execução de aplicações paralelas. Um exemplo desta arquitetura é GPU GeForce 280 da NVIDIA que possui 240 núcleos, cada qual com múltiplos *threads*, com um processamento de instruções simples que compartilham seu *cache* e controle de instruções.

O poder de processamento destas duas arquiteturas, tem evoluído de forma distinta, como pode ser observado na Figura 2.23. Os CPUs tiveram uma diminuição na sua taxa de crescimento, por conta de limitações físicas e de dissipação de calor, contudo a evolução dos GPUs não teve esta redução causando uma sensível distância quanto ao poder de processamento destas duas arquiteturas, o que em 2009 ficou claro com uma diferença de 10 para 1, em operações de ponto flutuante.

A evolução do GPU apresentou uma curva diferente do CPU, isso pode ser visto nos gráficos apresentados nas Figuras 2.23, que apresenta a diferença do poder de processamento de operações realizadas com ponto flutuante por segundo entre CPUs e GPUs, e a Figura 2.24 que apresenta o aumento da largura de banda para acesso a memória entre os mesmos dispositivos.

Essa diferença pode ser explicada principalmente pela forma como as arquiteturas dos dois

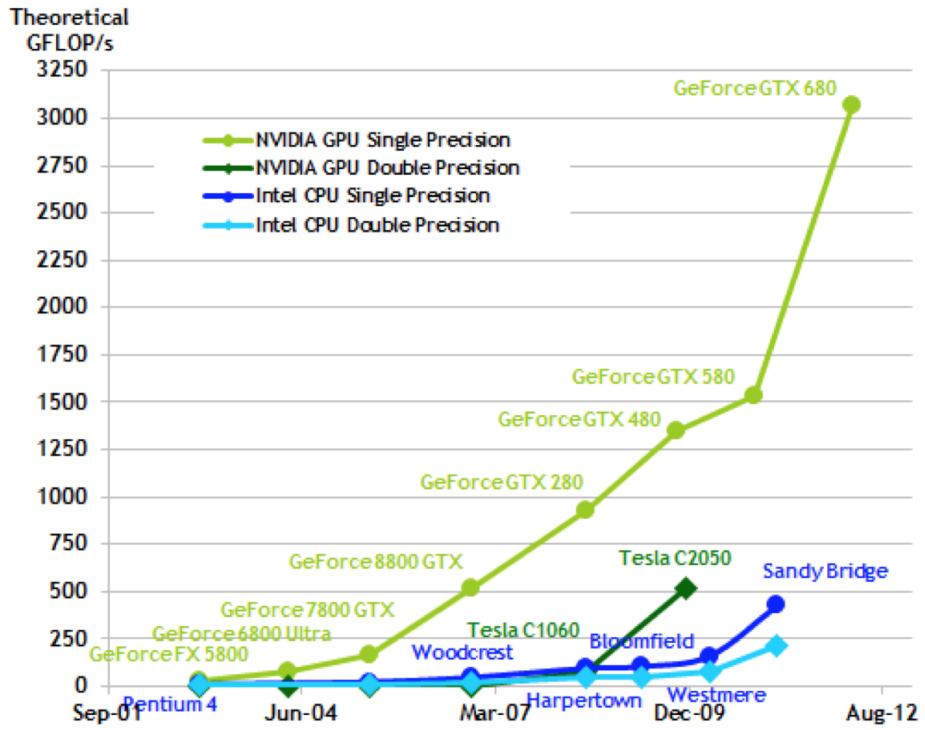


Figura 2.23: Comparação de operações de ponto flutuante entre CPU e GPU (NVIDIA, 2012).

dispositivos seguiram caminhos diferentes para a sua evolução. Como pode ser observado na Figura 2.25, a arquitetura do GPU (direita) possui um número muito maior de ALU (*arithmetic and logic unit*) do que o CPU (esquerda). Uma possível explicação dessa diferença evolutiva está no foco entre os dois dispositivos. Enquanto o CPU possui como foco o controle de processos mais complexos, incluindo tomadas de decisão, o GPU tem como principal foco a realização da mesma operação sobre uma grande massa de dados apresentando uma arquitetura do padrão *Single Instruction Multiple Threads*. Por outro lado, os GPUs apresentam um grande número de núcleos de processamento que compartilham o mesmo *cache* de dados e de instruções que são mais simples do que os apresentados nos CPUs. Desta forma uma mesma instrução pode ser executada em um número muito maior de dados.

Outra questão importante está na largura de banda de acesso dos núcleos de processamento para a memória. Os CPUs, por causa das exigências de sistemas legados e outros dispositivos que devem fazer acesso a *buffers*, como dispositivos de entrada e saída (I/O), a largura de banda da memória fica comprometida quanto à sua evolução, em contraste à memória mais simples e menos restritiva dos GPUs. Os GPUs apresentam uma largura de banda de memória muito maior que os CPUs. No final de 2006, o GeForce 8800 GTX era capaz de mover dados em cerca de 85 gigabytes por segundo (GB/s) dentro e fora de sua memória principal de acesso aleatório dinâmico (DRAM); já um modelo mais recente, a GT 200, suporta cerca de 150GB/s.

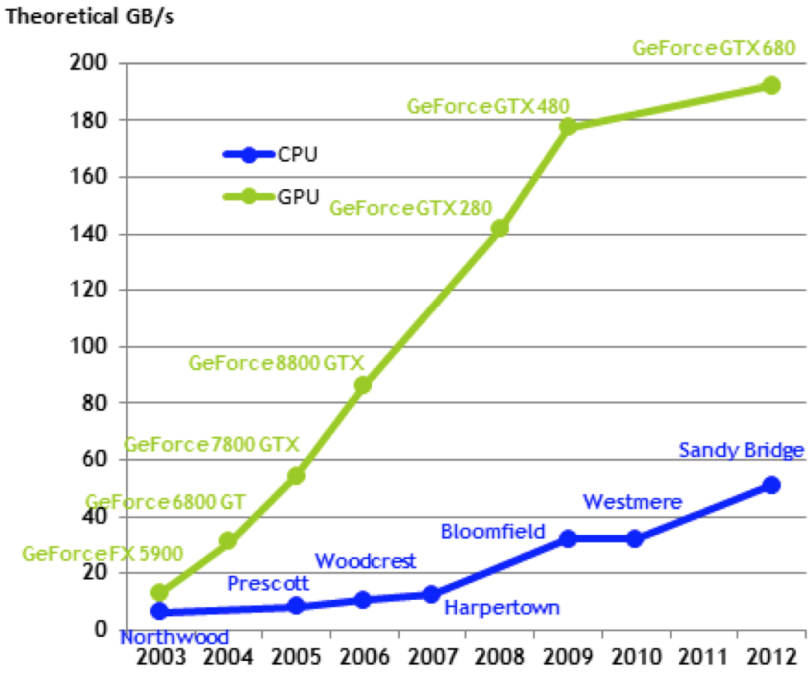


Figura 2.24: Comparação largura de banda para acesso a memória entre CPU e GPU (NVIDIA, 2012).

Figura 2.25: Diferença entre a arquitetura do CPU (esquerda) e GPU (direita) (NVIDIA, 2012).

Diante deste novo cenário, de processamento genérico usando o GPU, os fabricantes NVIDIA e ATI investiram no desenvolvimento de ferramentas com o objetivo facilitar o desenvolvimento de aplicações não gráficas, para este dispositivos. A NVIDIA lançou o CUDA (*Compute Unified Device Architecture*), que estende a linguagem de programação C, além de existirem *wrappers* que permitem a utilização de outras linguagens de programação, como Java (jCUDA), C# (CUDA.NET) e o Python (PyCUDA). Em resposta a ATI lançou o Brook+, baseado na linguagem de programação Brook, que também é uma variação da linguagem C.

Desta forma os GPUs foram concebidos como dispositivos de computação numérica, não possuindo o mesmo desempenho que o CPU em tarefas operacionais, mais complexas. Todo o projeto de computação que se destina à utilização destes dispositivos, deve levar em consideração a utilização tanto do CPU, quando do GPU, segundo as características de cada aplicativo

(POLI et al., 2007; POLI; SAITO; CECATTO, 2012).

Vale lembrar que paralelamente à NVIDIA a Intel tem desenvolvido arquiteturas *manycore* comparáveis, resultando no MIC (Many Integrated Core) coprocessador Xean Phi, lançado em 2012 (INTEL, 2013; SAULE; CATALYUREK, 2012). Os cores da arquitetura MIC são baseados numa versão modificada do P54C, usado originalmente no Pentium. O objetivo da arquitetura MIC é alavancar a herança do x86 criando uma arquitetura compatível com x86 que pode usar ferramentas de *software* de paralelização já existentes, incluindo OpenMP, OpenCL, Intel Cilk Plus e versões especializadas de Fortran, C++ e bibliotecas matemáticas.

2.12.2 CUDA - *Compute Unified Device Architecture*

Na Figura 2.26, é apresentado o fluxo de processamento para uma aplicação de um CPU que utiliza o GPU como um dispositivo de coprocessamento paralelo de dados. Neste ambiente existem dois tipos de código: o que será executado pelo CPU; e o que será executado pelo GPU. Desta forma, o primeiro passo (1) está em transferir os dados do CPU para o GPU, que é realizado por meio de uma cópia de uma dada área existente na memória principal para o espaço alocado na memória do GPU. Com os dados na memória do GPU, o CPU pode então fazer a chamada da função que será executada no GPU (2). Esta execução é realizada (3) por meio da criação de um estrutura que é organizada de forma hierárquica seguindo um Grid, que possui Blocos, onde estão alocadas as threads que consistem de uma sequência de instruções. No final do processamento os dados resultantes do processamento podem então serem copiados da memória do GPU, de volta para a memória do CPU (4).

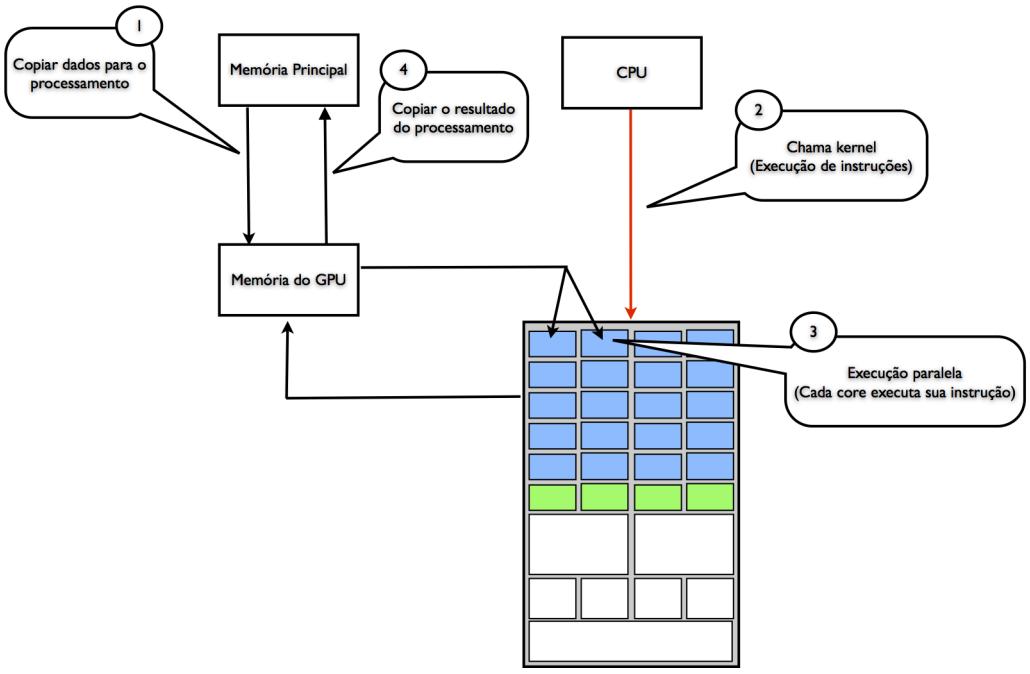


Figura 2.26: Fluxo de processamento para uma aplicação utilizando o CUDA.

Como um GPU permite o processamento utilizando um grande número de threads, uma aplicação CUDA organiza estes dentro de uma estrutura hierárquica, formada por três níveis de chamada que são: Grids, Blocos e Threads. Desta forma, quando um programa em execução no CPU, faz a chamada de uma função que irá ser executada no GPU este utiliza uma sintaxe especial do CUDA seguindo o formato:

nome_rotina_gpu«<tamanho_grid, tamanho_bloco»>(parâmetros_da_função)

As dimensões utilizadas na construção do Grid e do Bloco são particulares a cada aplicação, e definem o número de threads que serão utilizados. Na Figura 2.27(a) é apresentado um exemplo onde o CPU (Host) faz a chamada de uma função que será executada no GPU (Kernel 1), onde é utilizado um único Grid, formado por 6 Blocos, onde cada um destes possui 15 threads, o que cria um ambiente com 90 threads. No ambiente de execução de uma aplicação CUDA, existe um grupo de variáveis que descreve o ambiente no qual a função se encontra em execução. Neste exemplo estão sendo utilizados, o blockDim que informa o número de blocos de um grid, a blockIdx.x que informa a dimensão x de cada um dos blocos, e a threadIdx.x que informa a posição da thread dentro de um bloco. Estas variáveis são utilizadas para se obter o endereço de um determinado thread, de forma que se possa atribuir dados, como apresentado na Figura 2.27(b).

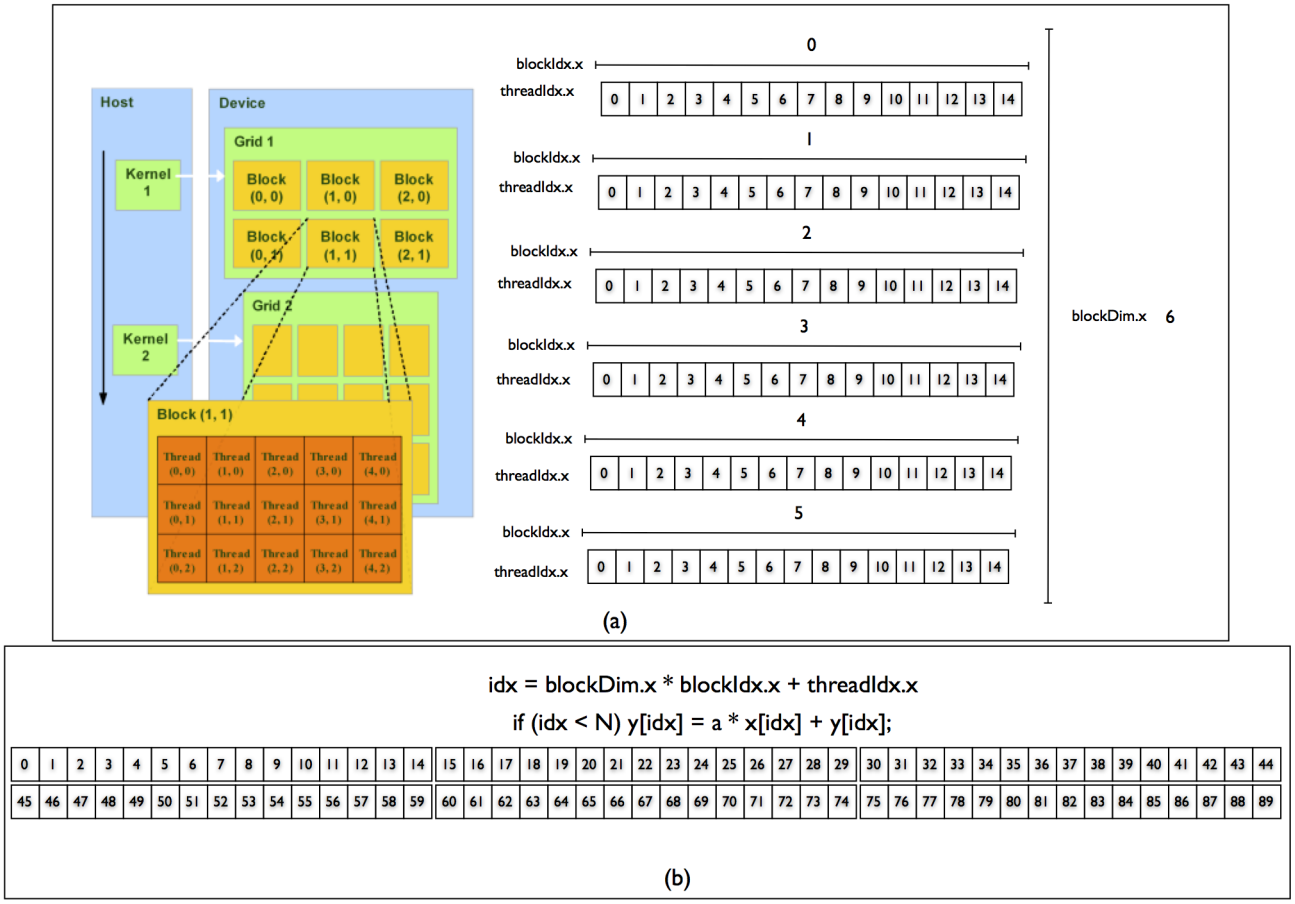


Figura 2.27: Estrutura Hierárquica do CUDA: Grids, Blocos e Threads.

A Figura 2.28 refere-se à arquitetura de um GPU da NVIDIA, que pode ser utilizado para o processamento de aplicações utilizando CUDA. Como pode ser observado, o GPU é formado por um arranjo de multiprocessadores de fluxo (*Streaming Multiprocessing* - SM), em destaque do lado direito, que são formados por um número de núcleos de processamento (*Streaming Processors* - SP) coloridos de verde que compartilham a mesma lógica de controle e o *cache* de instruções. Alguns modelos de GPU's, como GTX 290, possuem 4 Gb de DRAM dedicada, que é chamada de memória global do GPU. Esta memória DRAM difere da DRAM do CPU, existente na placa mãe, porque funciona como um *frame buffer* para aplicações gráficas, como vídeo, informações de textura e renderização de imagens em 3D. Entretanto, em aplicações genéricas estas se beneficiam da largura de banda existente, o que reduz em muito a latência de acesso existente no CPU. O GPU GTX 8800, que foi o primeiro modelo da NVIDIA com CUDA, lançado em 2006, possui 86,4 GB/s de largura de banda.

O CUDA fornece uma interface de comunicação que é utilizada pela aplicação em execução no CPU, que permite a cópia de blocos de dados da memória do CPU para a memória do GPU, e a chamada de instruções pelo CPU que serão executadas pelo GPU. Estas chamadas

de instruções são registradas em uma fila de execução que é responsável pelo escalonamento da execução das threads. O GPU modelo GTX 8800 possui 128 SPs, organizados em 16 SMs. Cada um destes SPs possui uma unidade *multiply-add* (MAD) e uma unidade de multiplicação adicional. Com estes 128 SPs é possível obter um desempenho correspondente a um total de 500 gigaflops. Já o modelo GT200 possui um total de 240 SPs o que permite exceder um total de 1 teraflops. Por causa da característica maciçamente paralela dos SPs, cada um destes pode executar centenas de *threads* por aplicação, onde um thread significa um conjunto sequencial de instruções de processamento. Em uma aplicação típica, cada SP pode executar cerca de 5.000 a 12.000 *threads* por segundo.

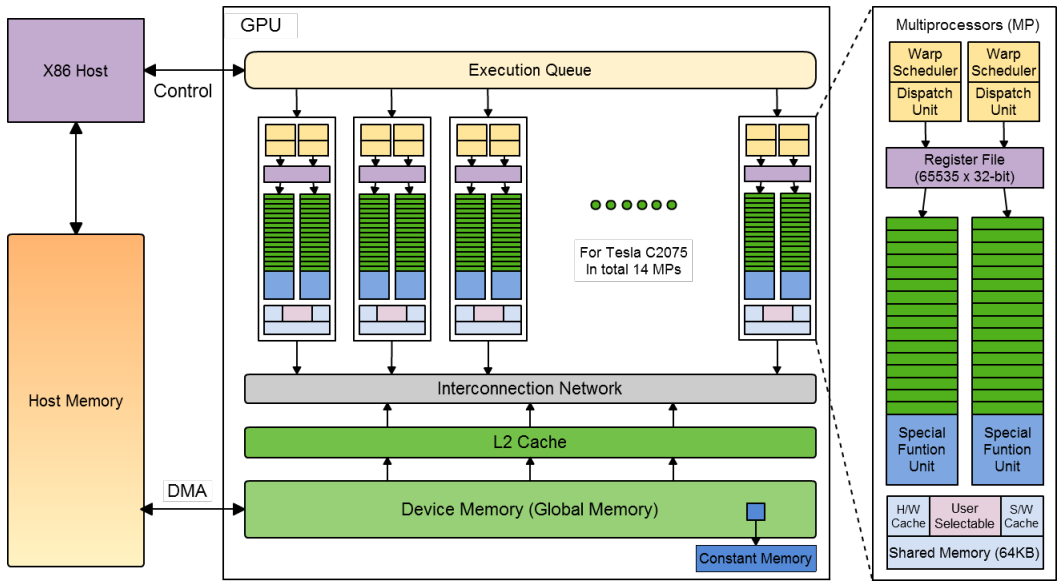


Figura 2.28: Arquitetura do GPU com o CUDA (CLUSTER, 2013).

Na Listagem 2.7, é mostrada uma implementação em C de uma função DAXPY, na qual são utilizados quatro parâmetros: *n* um inteiro que indica o número de elementos dos vetores *x* e *y*, *a* uma constante utilizada para multiplicar cada elemento do vetor *x* somado a cada elemento de *y*, e os próprios vetores *x* e *y*.

Algoritmo 2.7 Código em C para uma função DAXPY.

```
1 //Invoke DAXPY
2 daxpy(n, a, x, y);
3 void daxpy(int n, double a, double *x, double *y){
4         for (int i = 0; i < n; ++i)
5                 y[i] = a*x[i]+y[i];
6 }
```

A Listagem 2.8 corresponde à versão implementada em CUDA da Listagem 2.7. Nesta implementação é levado em consideração que serão utilizados 90 threads, estando estes orga-

nizados em 6 blocos, como ilustrado na Figura 2.28 . O processo de atribuição de valores às threads, apresentado na Figura 2.27, pode ser visto na linha 5 da Listagem 2.8.

Algoritmo 2.8 Versão implementada em CUDA do Algoritmo 2.7.

```
1 //Invoke DAXPY with 256 threads per Thread Block
2 daxpy<<6, 15>>>(n, 2.0, x, y);
3 __device__ void daxpy(int n, double a, double *x, double *y){
4         int i = blockIdx.x * blockDim.x + threadIdx.x;
5         if (i < n) y[i] = a * x[i] + y[i];
6 }
```

2.13 *Reconhecimento de Padrões e Near Sets*

O Reconhecimento de Padrões pode ser definido como sendo a área da ciência que trata da classificação e descrição de objetos, passando por três estágios, a extração de características dos objetos a serem classificados, a seleção das características que melhor representam os objetos de interesse, e a criação do classificador, ou descritor, capaz de ser utilizado para descrever objetos semelhantes (MARQUES, 2000).

Segundo o domínio do problema no qual os objetos de interesse se encontram inseridos, os estágios descritos podem ser implementados segundo diferentes abordagens, sendo as principais:

- Estatística: abordagem clássica, também conhecida como "Teoria da Decisão", que utiliza determinados modelos probabilísticos para o processo de reconhecimento dos padrões das classes;
- Sintática: que utiliza a estrutura dos padrões por meio da inter-relação das características descritivas básicas;
- Neural: que utiliza modelos inspirados em neurônios do cérebro para treinar o processo de reconhecimento; e
- Difusa: utiliza a teoria dos conjuntos difusos para modelar o grau de incerteza.

Para a aplicação implementada nesta tese foi usada a Teoria de Near Sets, descrita a seguir, que pode ser aproximada como uma abordagem sintática dentre as acima citadas.

A Teoria de NS (*Near Sets*) corresponde a uma ferramenta matemática que possui como objetivo a formalização de quão distantes, ou próximos, estão dois objetos. A similaridade

entre objetos é determinada pela comparação de valores presentes em cada um destes objetos, cuja comparação fornece a medida de quanto estes objetos se assemelham. Influenciada pelo trabalho sobre a teoria de *Rough Set* de Pawlak(PAWLAK, 2002), e de proximidade de conjuntos de Orlowska (ORLOWSKA, 1982), NS considera que dois objetos distintos apresentam algum tipo de similaridade se estes possuírem um conjunto de características comuns (HENRY, 2010; PETERS; WASILEWSKI, 2009), possuindo como definições:

Definição 1: Objeto - um objeto é algo que existe no mundo real e que possa ser descrito por meio de suas características.

Definição 2: Função de Descrição - que é uma função que retorna um valor real, $f : O \rightarrow \mathbb{R}$, e que representa uma característica do objeto, como exemplo tem-se a cor, textura, peso, entre outros. O conjunto de funções de descrição são responsáveis pela descrição de um objeto. O conjunto de todas funções de descrição é denotado por $\mathscr{F}$, e o conjunto específico de funções de descrição em uma aplicação é denotado por $B \subseteq \mathscr{F}$.

Definição 3: Sistema Perceptivo - consiste em um conjunto não vazio de objetos O junto com o conjunto de funções de descrição $\mathscr{F}$, sendo denotado por $< O, \mathscr{F} >$.

Definição 4: Descrição de um Objeto - assumindo um sistema perceptivo $< O, \mathscr{F} >$, a descrição de um objeto $x \in O$ é dada pelo vetor $\phi_B(x) = (\phi_1(x), \phi_2(x), ..., \phi_i(x), ..., \phi_l(x))$ onde l é o comprimento do vetor ϕ_B e cada $\phi_i \in B \subseteq \mathscr{F}$ é uma função de descrição de uma característica do objeto.

Definição 5: Relação de indiscernibilidade - assumindo um sistema perceptivo $< O, \mathscr{F} >$, para todo $B \subseteq \mathscr{F}$ a relação de indiscernibilidade entre dois objetos é definida como sendo $\sim_B = \{(x,y) \in O \times O : ||\phi_B(x) - \phi_B(y)||_2 = 0\}$, onde $||.||$ representa a norma L^2.

Definição 6: Relação de Tolerância - a relação de tolerância está associada ao grau de relaxamento que será aplicado à Definição 5, de forma que assumindo um sistema perceptivo $< O, \mathscr{F} >$, para todo $B \subseteq \mathscr{F}$ a relação de tolerância pode ser definida como

$\cong_{B,\varepsilon} = \left\{(x,y) \in O \times O : ||\phi_{B(x)} - \phi_B(y)||_2 \leq \varepsilon\right\}$, onde ε é a definição de tolerância a ser adotada.

Um exemplo do uso de NS pode ser visto na Tabela 2.3 onde é apresentado um conjunto de 20 (vinte) objetos com $|\phi(x_i)| \leqslant 1$. Assumindo $\varepsilon = 0.1$ como sendo o grau de tolerância entre as classes, ou seja $\|\phi_B(x) - \phi_B(y)\|_2 \leq \varepsilon$, o resultado da classificação (objetos próximos) pode ser visto na Figura 2.29.

x_i	$\phi(x)$	x_i	$\phi(x)$	x_i	$\phi(x)$	x_i	$\phi(x)$
x_1	.4518	x_6	.6943	x_{11}	.4002	x_{16}	.6079
x_2	.9166	x_7	.9246	x_{12}	.1910	x_{17}	.1869
x_3	.1398	x_8	.3537	x_{13}	.7476	x_{18}	.8489
x_4	.7972	x_9	.4722	x_{14}	.4990	x_{19}	.9170
x_5	.6281	x_{10}	.4523	x_{15}	.6289	x_{20}	.7143

Tabela 2.3: Exemplo de valores para um sistema perceptivo(HENRY; PETERS, 2012).

$$O=\left\{\begin{array}{l}\{x_1,x_8,x_{10},x_{11}\},\{x_1,x_9,x_{10},x_{11},x_{14}\},\\ \{x_2,x_7,x_{18},x_{19}\},\{x_3,x_{12},x_{17}\},\\ \{x_4,x_{13},x_{20}\},\{x_4,x_{18}\},\{x_6,x_{13},x_{20}\},\\ \{x_5,x_6,x_{15},x_{16}\},\{x_5,x_6,x_{15},x_{20}\}\end{array}\right\}$$

Figura 2.29: Classificação dos objetos apresentados na Tabela 2.3, como sendo próximos baseado na Definição 6(HENRY; PETERS, 2012).

2.14 GLCM - *Gray Level Co-occurence Matrix*

GLCM pode ser definido como um método estatístico para realizar a classificação da imagem utilizando as características de sua textura definida por Haralick e seus colaboradores na década de 1970 (HARALICK; SHANMUGAN, 1973). Por meio desta abordagem, utilizando uma imagem em tons de cinza, pode-se utilizar a relação de vizinhança dos pixels para se criar uma matriz GLCM sobre a qual pode-se extrair um conjunto de informações sobre uma determinada região da imagem.

Tom de cinza e a textura são duas propriedades sempre presentes na imagem e possuem uma relação muito próxima uma com a outra. Quando em uma pequena região da imagem existe uma pequena variação nos tons discretizados de cinza, refere-se a um atributo no domínio do tom de cinza da imagem. Contudo quando se trata da uma variação maior sobre uma superfície/área da imagem refere-se a um atributo no domínio da textura da imagem. O ponto principal da divisão de tom e textura está no tamanho da área de análise destas variações e o número/quantidade de variações ocorridas (HARALICK; SHANMUGAN, 1973).

Assumindo-se que a imagem a ser processada seja retangular e a sua resolução horizontal seja indicada por N_x e a a vertical por N_y, e que os tons de cinza utilizados nesta imagem tenham sido quantizados em N_g níveis, desta forma $L_x = \{1,2,...,N_x\}$ representa o espaço horizontal e $L_y = \{1,2,...,N_y\}$ o espaço vertical e $G = \{1,2,...,N_g\}$ os valores de tons de cinza existentes na imagem. Desta forma uma imagem I pode ser representada como sendo $I : L_x \times L_y \rightarrow G$. Um

componente essencial do conceito da textura é a medida, mais precisamente das quatro medidas de sua vizinhança, que são determinadas pela vizinhança angular dos tons de cinza, Na Figura 2.30, pode ser observado que a posição de um pixel $*$ possui quatro vizinhos que são indicados pelos ângulos 0, 45, 90 e 135 graus, sendo os vizinhos indicados pelos pares 1 e 5, 4 e 8, 3 e 7, 2 e 6 respectivamente.

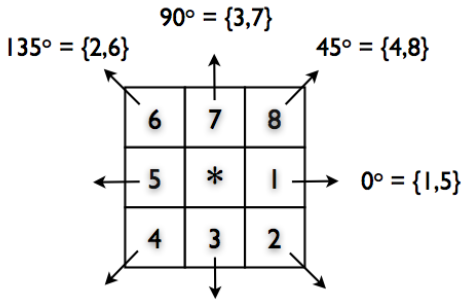


Figura 2.30: Vizinhança angular dos tons de cinza de um pixel.

É assumido que o contexto de informação da textura em uma imagem I contém a média espacial do relacionamento de todos os tons de cinza desta imagem, o que faz com que esta informação seja adequada para especificar uma matriz com as frequências relativas P_{ij} entre duas células vizinhas separadas por uma distância d, onde i e j indicam os tons de cinza destes vizinhos. Desta forma pode-se definir como matriz de tons de cinza da dependência espacial como uma função do dependência dos ângulos da vizinhança das células e da distância. Na Figura 2.31 é apresentado um exemplo do processo da obtenção das matrizes de GLCM, onde a Figura 2.31(a) é a matriz de uma imagem 4x4 que possui quatro tons de cinza e a Figura 2.31(b) é a matriz quadrada de coordenadas formada pelos tons de cinza da imagem de entrada, e as demais matrizes são os resultados do cálculo da ocorrência de tons de cinza na vizinhança segundo a direção odotada 0, 45, 90, 135 graus, ou sejam as matrizes GLCM nas respectivas direções.

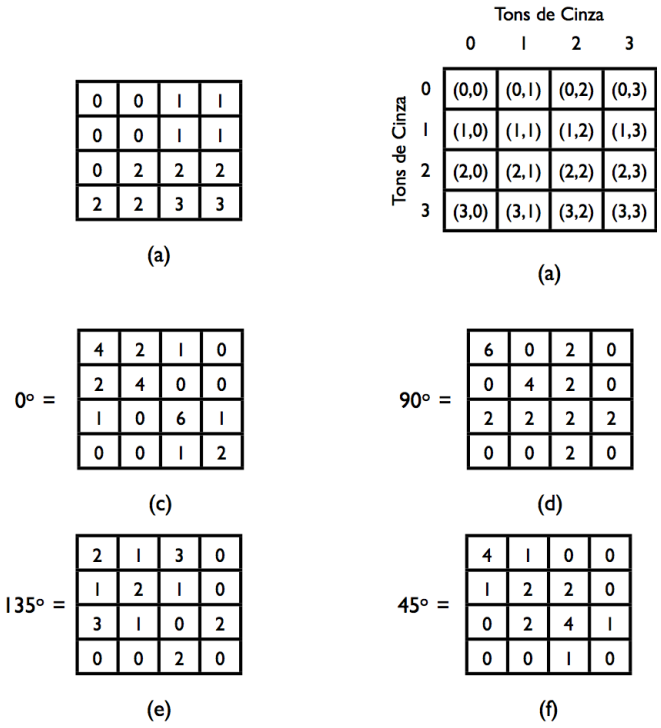


Figura 2.31: Cálculo dos valores da matriz.

Como exemplo será calculada a Matrix GLCM com 0^o de orientação (Figura 2.31(c)). Seguindo a indicação da Figura 2.30, 0^opossui como orientação para suas vizinhas as posições no conjunto {1,5} que indicam pixel à direita e pixel à esquerda, respectivamente. Desta forma, tomando o primeiro elemento da matriz de coordenadas de tons de cinza (Figura 2.31(b)), a coordenada (0,0), interpreta-se: sendo a posição da matriz ocupada por um tom de cinza de valor 0, quantas posições possuem 0 à sua esquerda ou à direita, percorrendo todas as coordenadas da imagem da Figura 2.31(a). Nota-se que a resposta é 4 como mostrado na Figura 2.31(c).

Os demais elementos da Figura 2.31(c) são obtidos similarmente, levando-se em conta que os elementos da Figura 2.31(b), devem ser interpretados como que o elemento à esquerda seja o valor do pixel da matriz de entrada 4x4 e o elemento à direita é o valor do pixel vizinho na direção considerada, segundo a orientação indicada na Figura2.30.

O mesmo raciocínio utilizado para o cálculo da Matrix GLCM de 0^o(Figura 2.31(c)) pode ser adotado para as demais orientações de vizinhança, utilizando suas respectivas coordenadas. Neste trabalho foram utilizados como funções de descrição o Contraste, a Correlação, a Homogeneidade, a Energia e a Entropia das matrizes de GLCM, cujas equações são apresentadas na Tabela 2.4.

Função	Equação
Contraste	$\sum_{N-1}^{i,j=0} P_{i,j}(i-j)^2$
Correlação	$\sum_{i,j=0}^{N-1} P_{i,j} \left[\frac{(i-\mu_i)(j-\mu_j)}{\sqrt{(\sigma_i^2)(\sigma_j^2)}} \right]$
Energia	$\sum_{i,j=0}^{N-1} P_{ij}^2$
Entropia	$\sum_{i,j=0}^{N-1} P_{i,j} \left(-lnP_{i,j}\right)$
Homogeneidade	$\sum_{i,j=0}^{N-1} \frac{P_{i,j}}{i+(i-j)^2}$

Tabela 2.4: Equações das funções de descrição: Contraste, Correlação, Energia, Entropia, e Homogeneidade.

2.15 Interferometria e Síntese de Abertura

Nesta seção é apresentada a base teórica de um Radiointerferômetro, instrumento utilizado para a observação de eventos cósmicos na banda de rádio frequência, por meio da correlação dos sinais obtidos por um conjunto de antenas organizadas em um arranjo, conhecido como um arranjo interferométrico.

A radioastronomia pode ser entendida como sendo uma sub-área da astronomia em que as observações são feitas na banda de rádio frequência utilizando para isso um instrumento chamado Radiotelescópio. Sua utilização é de grande importância pelas características que podem ser observadas no Espectro Eletromagnético, apresentado na Figura 2.32, onde nem todas as frequências apresentadas chegam à superfície da Terra. Para as frequências mais baixas, a ionosfera absorve a radiação enquanto que para as ondas de comprimento em mm e sub-mm é o vapor d'água atmosférico que as absorve. E é justamente esta restrição que faz com que a radioastronomia seja importante, pois a faixa de frequência das ondas de rádio chegam à superfície da Terra sem grandes perdas, o que permite a construção de instrumentos para a sua captação e estudo.

A principal diferença entre o radiotelescópio e o telescópio óptico, é que o radiotelescópio possui um sistema coerente, com a preservação da informação de fase do sinal. Desta forma, o mesmo é construído segundo o comprimento de onda das fontes de interesse, que vão desde alguns milímetros até a ordem de quilômetros. A maioria dos radiotelescópios possuem uma antena parabólica (em forma de prato) que funciona como refletores que podem ser apontados para qualquer parte do céu. De forma simplificada, estes instrumentos são formados pelos seguintes elementos: 1) uma antena, que pode ser descrita como um conjunto formado por um refletor, responsável por coletar a radiação eletromagnética de uma determinada região do espaço, que a concentra e envia ao alimentador, que é responsável por transformar a radiação

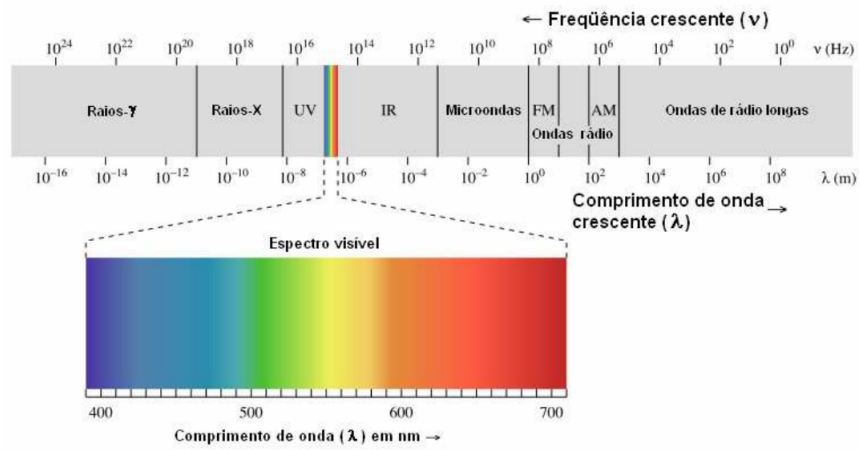


Figura 2.32: Espectro eletromagnético da Terra (CECATTO, 2011).

incidente em sinais elétricos correspondentes; 2) um receptor, que é responsável pela seleção e amplificação do sinal proveniente do alimentador; e 3) um sistema para o registro do sinal observado, onde os dados são registrados para análise futura.

A resolução angular de um radiotelescópio representa a menor distância angular na qual duas fontes pontuais podem ser distintas. Usualmente, a resolução angular de uma determinada antena é definida na largura à meia potência do lóbulo principal (*Half Power Beam Width*, HPBW), a qual corresponde à extensão angular entre os dois extremos do lóbulo principal onde a potência normalizada vale $1/2$. A resolução angular δ de um radiotelescópio é dada em função do diâmetro da antena D e do comprimento de onda observado λ (AVISON; GEORGE, 2013) conforme Equação 2.3.

$$\delta \approx \frac{\lambda}{D} \tag{2.3}$$

Pode-se observar que antenas maiores apresentam uma melhor resolução angular, uma vez que são capazes de coletar uma maior potência eletromagnética. Um outro fator no qual influencia a dimensão de uma antena, é que antenas maiores possuem uma melhor capacidade de discriminar sinais provenientes de diferentes direções, ou seja, quanto maior o diâmetro de uma antena mais precisa será esta discriminação (resolução espacial). A ordem de grandeza do comprimento de onda observado em rádio frequência, faz com que a resolução espacial destes instrumentos sejam inferiores quando comparados com os instrumentos ópticos, onde o comprimento de onda observado é da ordem de centenas de vezes menores que as ondas de rádio, de forma que, para se obter uma melhor resolução espacial, é necessário um instrumento com uma antena de grande diâmetro.

Contudo um radiotelescópio, com uma antena de diâmetro muito grande, pode ser inviável,

uma vez que exige uma estrutura mecânica muito grande. Como exemplo pode-se destacar os radiotelescópios de Arecibo (Figura 2.33), onde, pela exigência do diâmetro de sua antena, cerca de 305 metros, este foi construído dentro de uma depressão em Porto Rico (ARECIBO, 2011).

Figura 2.33: Foto do radiotelescópio de Arecibo em Porto Rico (PLANET, 2010).

Entretanto, pode-se realizar observações radioastronômicas com boa resolução espacial por meio de um radiointerferômetro, que pode ser definido como uma combinação de radiotelescópios, observando uma mesma fonte, de forma a aumentar a resolução espacial, por meio da correlação de seus sinais, o que produz uma medida conhecida como função de coerência mútua ou função visibilidade (WILSON; ROHLFS; HÜTTEMEISTER, 2009), que corresponde à medida de um componente espacial de Fourier da distribuição de brilho da fonte observada (Figura 2.34), desta forma, arranjos radiointerferométricos são basicamente conjuntos de elementos (antenas) organizados em pares, que são usados para fazer medições em detalhes de fontes de rádio no céu.

Em um radiointerferômetro, a distância entre duas antenas, faz o papel de diâmetro na Equação 2.3, isso porque, a resolução espacial obtida pelas antenas, observando a mesma fonte, é inversamente proporcional à distância entre elas. Já o número de antenas que formam o arranjo interferométrico, é responsável pela qualidade da cobertura do plano de amostragem.

2.15.1 Correlacionador de duas antenas

Na Figura 2.35 é apresentado um radiointerferômetro formado por duas antenas, onde b é o vetor unitário de linha de base, s é um vetor unitário que aponta para a fonte que está sendo

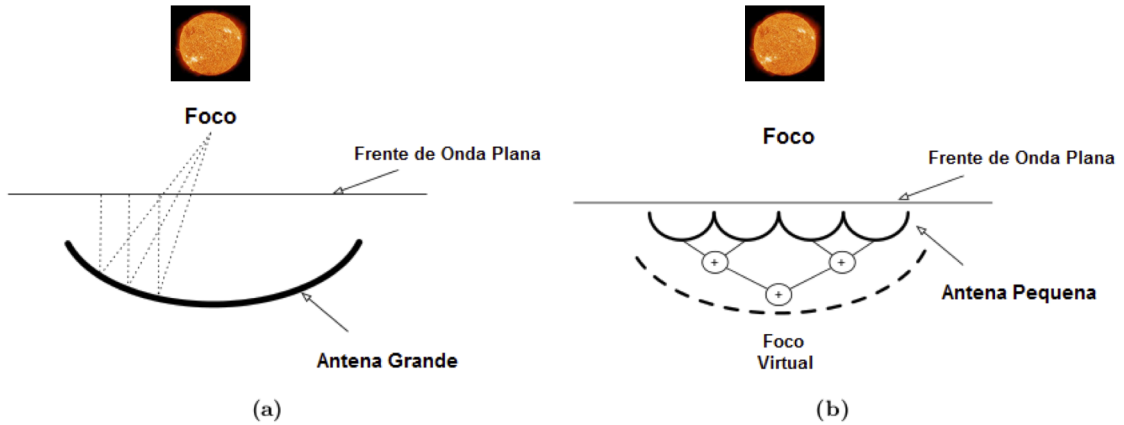


Figura 2.34: Combinação de dois ou mais radiotelescópios de forma a criar uma arranjo interferométrico, para aumentar a resolução angular.

observada, e τ_g o atraso geométrico em uma das antenas. Na parte inferior da figura encontra-se em destaque o correlacionador.

As operações básicas de um correlacionador são a multiplicação e a integração. Os sinais recebidos de cada antena passam por um circuito correlacionador, por um determinado período de tempo, produzindo uma medida entre os sinais, que pode ser uma interferência construtiva ou destrutiva. Supondo que os sinais percorram uma mesma distância através dos cabos, então o plano de onda incidente nas antenas A_1 e A_2 estarão em fase se o ângulo θ formado entre a direção de propagação do plano de onda e a normal do vetor de linha de base for zero (fonte localizada no zênite). Desta forma a resposta máxima será obtida pela correlação cruzada entre os sinais.

Entretanto, se a fonte for deslocada do zênite por um ângulo pequeno θ, haverá uma pequena diferença de percurso $\delta 1 = b \sin\theta$ entre os sinais que chegam às antenas A_1 e A_2 , implicando em uma diferença de fase entre eles na entrada do correlacionador (FARIA, 2006). Esta diferença de fase entre os sinais produzirá um valor de medida de correlação entre os sinais um pouco menor, podendo até mesmo assumir um valor nulo quando a diferença de fase for máxima, ou seja, 180^o.

Para diferenças de percurso δl cujo valor seja múltiplo inteiro do comprimento de onda da fonte observada, os sinais que chegam até o correlacionador estarão em fase novamente, o que produz um máximo da medida de correlação. Esta variação da intensidade das medidas de correlação, em função deδl geram as franjas interferométricas que podem ser observadas na Figura 2.36. O período de uma franja interferométrica é definido pela distância angular Θ entre os máximos consecutivos da franja:

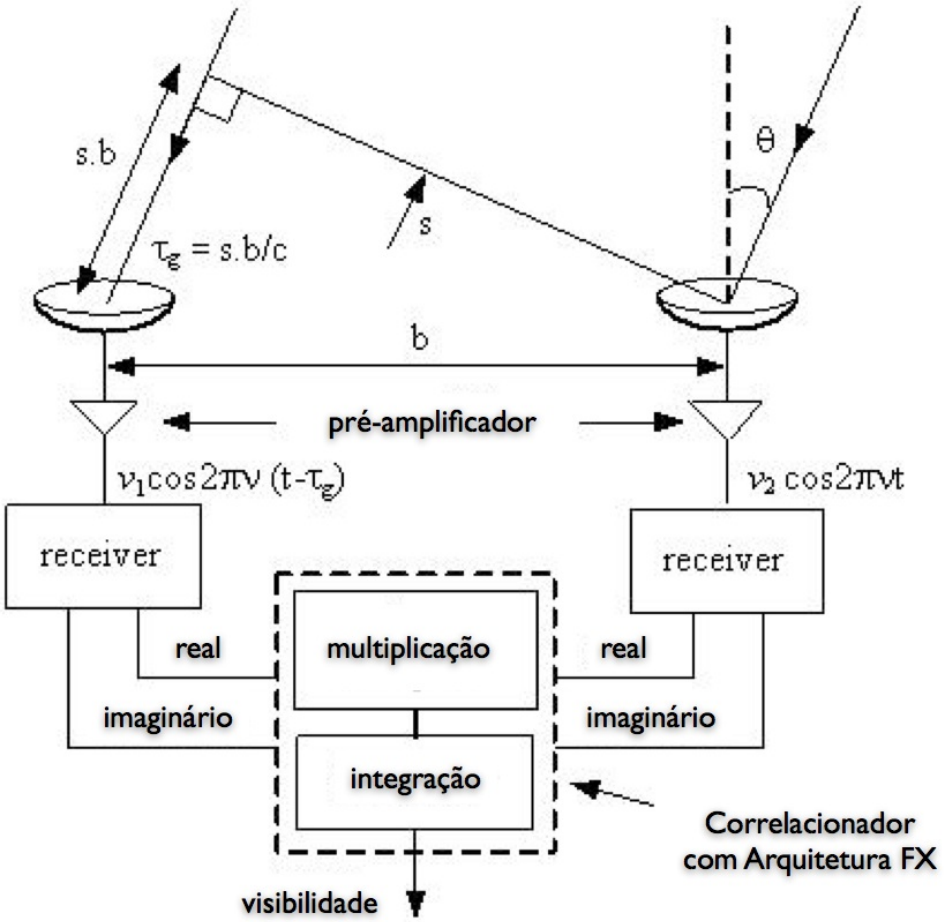


Figura 2.35: Exemplo de um correlacionador com duas antenas.

$$\Theta = \sin^{-1}\left(\frac{\lambda}{b_{proj}}\right) \tag{2.4}$$

onde b_{proj} é a linha de base projetada no plano da fonte, ou seja, no plano normal à direção de propagação da onda, sendo que:

$$b_{proj} = b\cos\theta \tag{2.5}$$

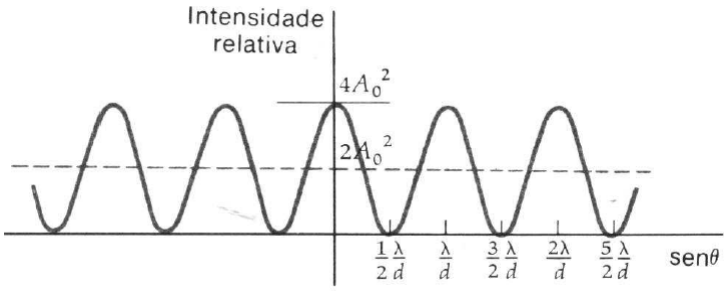


Figura 2.36: Exemplo de franjas radiointerferométricas.

Se Θ for um ângulo relativamente pequeno então pode-se aproximar a Equação 2.4 tal que o período Θ da franja interferométrica seja dado por:

$$\Theta \approx \sin^{-1}\Theta = \frac{\lambda}{b_{proj}} \tag{2.6}$$

Desta forma a frequência espacial $u = \Theta^{-1}$ da franja interferométrica é dada por:

$$u = \frac{b_{proj}}{\lambda} \tag{2.7}$$

Desta forma um radiointerferômetro mede a função de coerência mútua de um campo elétrico, que tem como origem a distribuição de brilho no céu de uma fonte de rádio. As antenas de um radiointerferômetro, convertem o sinal deste campo elétrico em voltagem, e a medida de função de coerência mútua é obtida por meio da correlação cruzada da voltagem de cada par de antenas.

O resultado obtido pela correlação cruzada dos sinais das antenas é chamado visibilidade. O dispositivo utilizado para medir a visibilidade espectral é chamado de correlacionador complexo. Estes dispositivos são implementados utilizando técnicas de processamento digital de sinais. A visibilidade medida por um radiointerferômetro é caracterizada pela amplitude e fase da franja em diferentes instantes. A saída do correlacionador de duas antenas pode ser escrita como sendo:

$$r_R(\tau_g) = Re\left[v_1(\nu,t)v_2^*(\nu,t)\right] = \Gamma\cos(2\pi\nu\tau_g + \Phi_\nu) \tag{2.8}$$

onde $v_i(\nu,t)$ e $v_2^*(\nu,t)$ são as voltagens de saída das antenas 1 e 2 (entradas do correlacionador), Γ e Φ são respectivamente a amplitude e a fase da visibilidade e τ_g é o atraso geométrico. As quantidades necessárias para fazer o mapeamento de um objeto celeste são Γ e Φ de todos os pares de antenas do radiointerferômetro.

2.15.2 Atraso Geométrico

Os sinais recebidos pelas antenas são convertidos para uma banda base e misturada com o sinal de oscilador local de frequência ν_{LO}. A compensação do atraso geométrico é formada introduzindo atrasos no sinal de banda base. A saída do correlacionador, depois da introdução do atraso τ_i, pode ser escrita como:

$$r_R(\tau_g) = \Gamma\cos(2\pi\nu\tau_g - 2\pi\nu_{BB}\tau_i + \Phi_\nu) = \Gamma\cos(2\pi\nu_{LO}\tau_g - 2\pi\nu_{BB}\Delta\tau_i + \Phi_\nu) \tag{2.9}$$

onde ν_{BB} é a frequência da banda base e $\Delta\tau_i = \tau_g - \tau_i$ é o atraso residual. Existem dois termos que surgiram nesta equação devido a compensação do atraso: $2\pi\nu_{BB}\Delta\tau_i$ e $2\pi\nu_{LO}\tau_g$. O primeiro termo é devido à precisão finita da compensação do atraso e o segundo é consequência do atraso ser compensado na banda base. A fase $2\pi\nu_{BB}\Delta\tau_i$ depende de ν_{BB}. Este termo, produz um gradiente de fase com uma largura de banda $\Delta\nu$. O gradiente de fase é uma função das mudanças do atraso no tempo. A fase $2\pi\nu_{LO}\tau_g$ é independente de ν_{BB}, sendo constante em toda a banda. Esta fase também é função do tempo devido à dependência de τ_g. Portanto, ambas as quantidades devem ser compensadas dinamicamente (ROSHI, 2003).

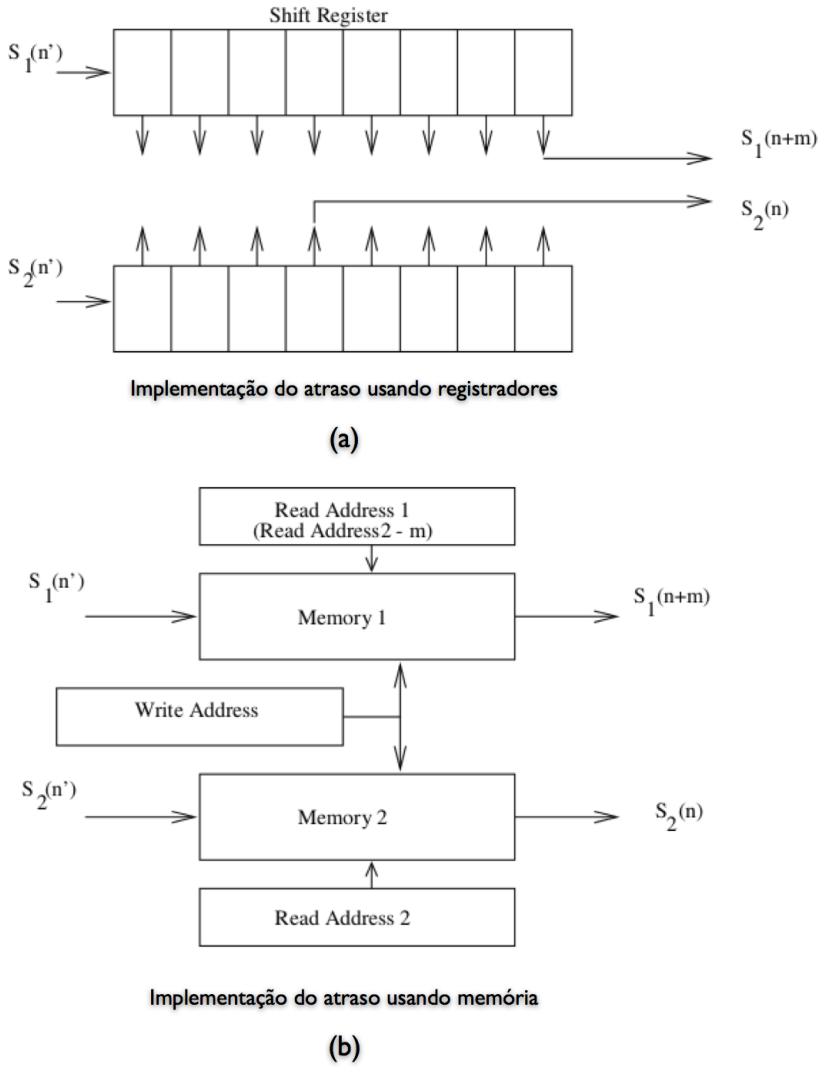


Figura 2.37: Implementação do atraso, na parte superior, usando registradores de deslocamento e, em baixo, usando memória (ROSHI, 2009).

A compensação do atraso em múltiplos intervalos de amostras de $1/f_s$ pode ser obtido por meio de deslocamentos dos dados amostrados, como apresentada na Figura 2.37(a). As amostras digitalizadas passam por registradores de deslocamento, onde o tamanho destes registradores é utilizado para inserir o atraso entre estes sinais.

Um outra técnica de inserir o atraso entre os sinais é utilizando memória de acesso aleatório, como pode ser visto na Figura 2.37(b). Neste modelo, os dados das antenas são escritos na memória RAM, para que então, no momento de sua saída, por meio de utilização de ponteiros, possa ser adicionado este atraso.

Um atraso fracionário pode ser introduzido mudando a fase do relógio de amostragem. A fase é alterada tal que sinais de suas antenas sejam amostrados com uma diferença de tempo igual ao atraso fracionário. Um segundo método é introduzir gradientes de fase no espectro do sinal. Esse gradiente de fase pode ser introduzido após a transformada de Fourier dos sinais das antenas.

Para se obter a informação de fase e de amplitude do sinal do sinal amostrado, é necessário a utilização de um correlacionador mais elaborado conhecido como correlacionador complexo, onde o sinal obtido pelas antenas passa por uma Transformada de Hilbert, causando um deslocamento de 90^o na fase do sinal. Em um correlacionador complexo os dois sinais de voltagem produzidos pelas antenas $V_1(t)$ e $V_2(t)$ é dado pela Equação 2.10, onde ν é a frequência do sinal, t o tempo, A_1 e A_2 as amplitudes, e α_1 e α_2 as fases dos dois sinais respectivamente.

$$V_1(t) = A_1 e^{j(2\pi\nu t + a_1)} \qquad V_2(t) = A_2 e^{j(2\pi\nu t + \alpha_2)} \tag{2.10}$$

A saída de um correlacionador complexo é dada pela média temporal do produto da multiplicação do sinal de entrada da primeira antena $V_1(t)$ pelao complexo conjugado da segunda $V_2(t)$ como mostrado na Equação 2.11.

$$\begin{aligned} \langle V_1(t)V_2^*(t)\rangle &= A_1 e^{j(2\pi\nu t+\alpha_1)} \times A_2 e^{-j(2\pi\nu t+\alpha_2)} \\ &= A_1A_2\left[\cos(\alpha_1-\alpha_2) + j\sin(\alpha_1-\alpha_2)\right] \\ &= A_1A_2\left[\cos(\alpha_1-\alpha_2) + j\cos\left(\alpha_1+\tfrac{\pi}{2}-\alpha_2\right)\right] \end{aligned} \tag{2.11}$$

Como apresentado na Equação 2.11, um correlacionador complexo tem como resultado um número complexo, composto pela parte real dado por $A_1A_2\cos(\alpha_1-\alpha_2)$ e pela parte imaginária

dado por $A_1 A_2 j \cos(\alpha_1 + \pi/2 - \alpha_2)$.

2.15.3 Arquitetura de Correlacionadores: FX e XF

Basicamente um correlacionador complexo realiza duas operações: uma transformada de Fourier (F) e uma multiplicação (X). A ordem com a qual estas duas operações são realizadas determina a arquitetura do correlacionador, em correlacionador FX ou correlacionador XF. O sinal, com uma banda limitada, pode ser decomposto em componentes espectrais por meio do uso de banco de filtros. A visibilidade espectral pode ser obtida pelo cruzamento, ou multiplicação (X), da saída de cada filtro, utilizando-se para isso um correlacionador complexo, como o apresentado no diagrama de blocos da Figura 2.38.

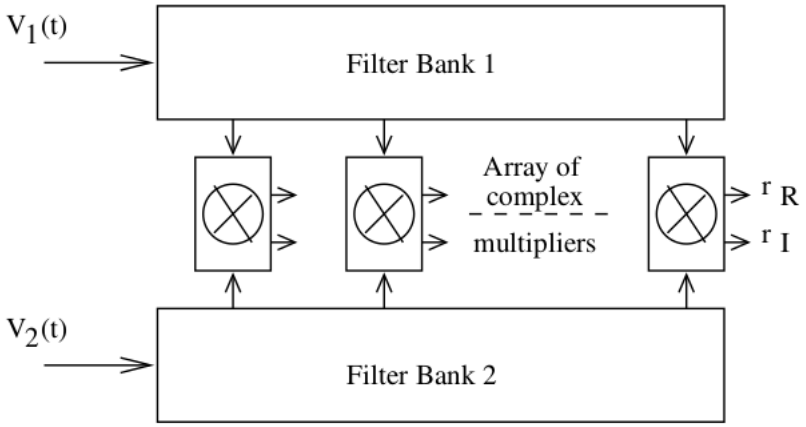


Figura 2.38: Diagrama de blocos de um correlacionador complexo(ROSHI, 2009).

Um correlacionador FX, como o apresentado na Figura 2.39, executa uma Transformada de Fourier antes de fazer a multiplicação. As voltagens analógicas, $V_1(t)$ e $V_2(t)$, são digitalizadas utilizando conversores ADCs. Em seguida os atrasos geométricos são inseridos no sinal, nos intervalos amostrados. O atraso integral compensado das amostras são multiplicados pela saída do NCO (*Number Controlled Oscillator*) para evitar franjas interferométricas. As amostras de cada antena são processadas por um bloco de FFT, para então ser aplicado o gradiente de fase, de forma a obter o atraso fracionado.

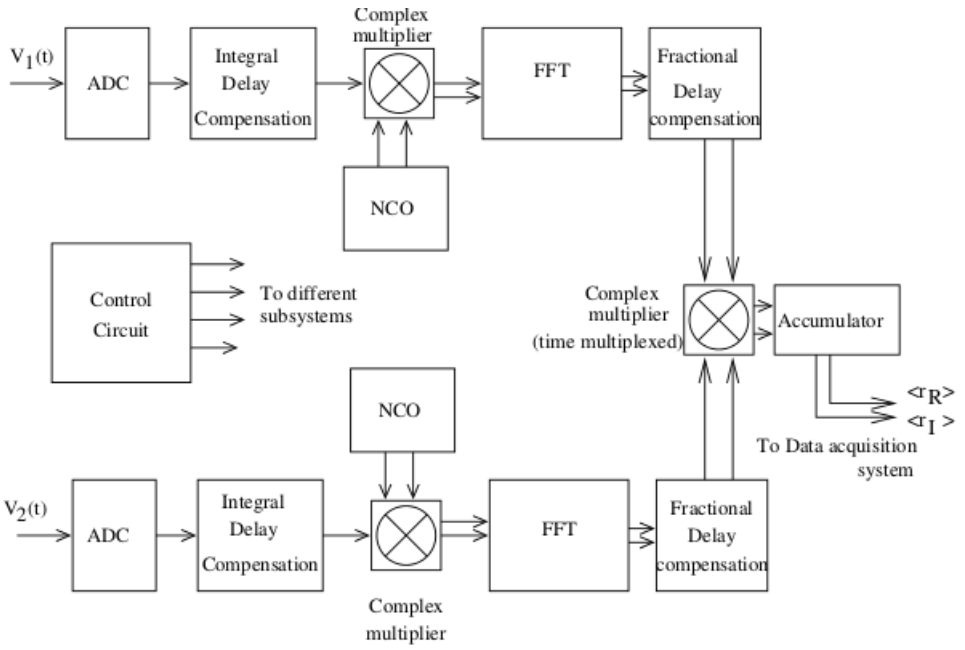


Figura 2.39: Diagrama de blocos de um correlacionador FX (ROSHI, 2009).

A visibilidade espectral é obtida pela multiplicação dos componentes espectrais de uma antena com os correspondentes componentes das demais antenas, que então são integrados, dentro de um intervalo de tempo determinado, para se obter uma estimativa da correlação cruzada.

Um diagrama de blocos do correlacionador XF pode ser visto na Figura 2.40. Neste diagrama os atrasos fracionados são compensados pela mudança de fase na taxa de amostragem. Depois da compensação do atraso, a correlação cruzada dos diferentes atrasos é medida usando linhas de atraso e multiplicadores, que são seguidos de um integrador. Uma vez que a função de correlação cruzada não é uma função par de τ, a compensação do atraso é feita de tal forma que a função de correlação é medida por ambos os valores, positivos e negativos de τ no correlacionador. A correção da quantização, bloco marcado como F na Figura2.40, é aplicada para normalizar a correlação cruzada. O espectro do cruzamento é obtido pelo bloco de FFT.

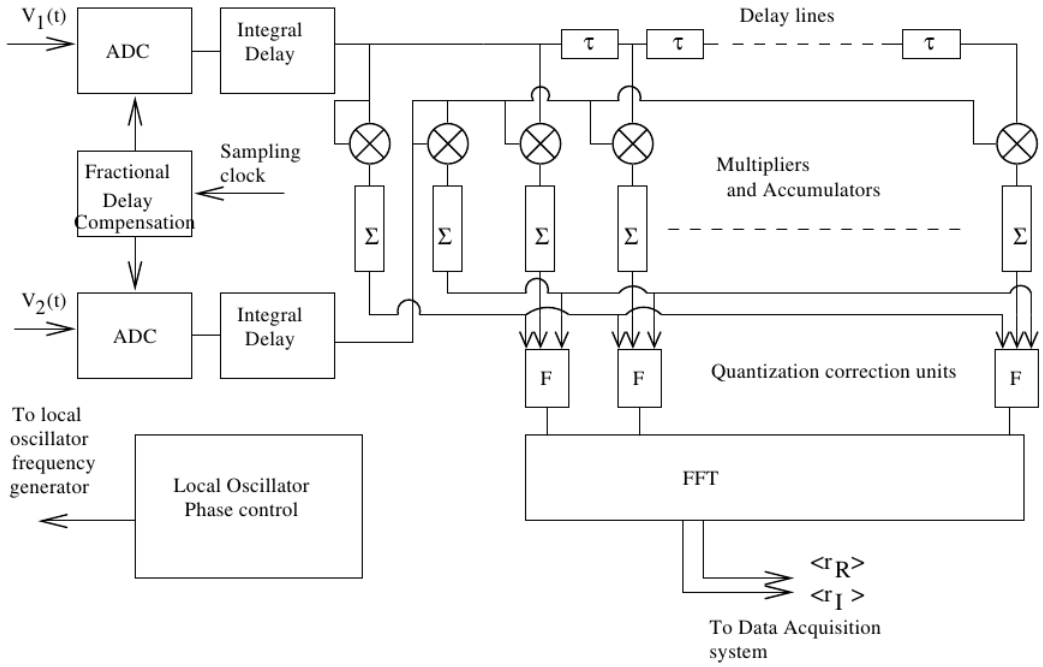


Figura 2.40: Diagrama de blocos de um correlacionador XF (ROSHI, 2009).

O custo computacional de um correlacionador, seja nas operações FFT ou na integração do sinal, depende do número de canais e de linhas de base do arranjo (Tabela 2.5). O custo da execução do FFT cresce a uma taxa de $O(N_c \log N_c)$, onde N_c representa o número de canais por antena, e N_a o número de antenas do arranjo. O número de linhas de base N_b cresce quadraticamente ao número de antenas do arranjo, ou seja;

$$N_b = \frac{(N_a - 1)(N_a)}{2} \tag{2.12}$$

Portanto, o custo da fase de integração cresce a uma taxa $O(N_a^2)$ em relação ao crescimento do número de antenas e linearmente ao número de canais.

	FFT	Integração
Computacional	$N_c \log N_c \times N_a$	$\frac{(N_a-1)(N_a)}{2} \times N_c$
Outros	$O(N \log N)$	$O(N^2)$

Tabela 2.5: Custo computacional das operações do correlacionador (WOODS, 2010).

Na Tabela 2.6 são apresentadas as características de dois instrumentos, o meerKAT, na África, e o SKA, na Austrália, (em 2012 e 2021), e para cada um destes as exigências do poder computacional exigido para atender a suas demandas (WOODS, 2010).

	meerKAT	SKA (2017)	SKA (2021)
Antenas	80	620	2400
Taxa de dados (por antena)	32 Gbps	32 Gbps	32 Gbps
Taxa de dados (total)	3 Tbps	20 Tbps	80 Tbps
Poder computacional requerido	52 TeraOps	3 PentaOps	47 PetaOps

Tabela 2.6: Custo computacional das operações do correlacionador (WOODS, 2010).

2.16 BDA - Brazilian Decimetric Array

Nesta seção é descrito o Brazilian Decimetric Array - BDA, um radiointerferômetro em construção pelo INPE, tendo o monitoramento de atividades solares, como função principal, e que serviu de inspiração para a presente tese de doutorado. O BDA está sendo construído em três fases distintas, sendo que cada fase apresenta um requisito específico de projeto. Na primeira fase de desenvolvimento, um protótipo do arranjo foi colocado em operação no campus do INPE em São José dos Campos em 2003 (FARIA, 2006) para testes dos subsistemas mecânicos e eletrônicos.

O arranjo implementado era composto por 5 antenas regularmente distribuídas em uma única linha de base na direção Leste-Oeste, utilizando-se um espaçamento fundamental de 8 metros entre as antenas, com uma máxima linha de base de 32 metros (FARIA, 2006) (Figura 2.41).

Figura 2.41: Foto do protótipo do BDA composto por 5 antenas instalado no campus do INPE em São José dos Campos.

Em 2004, após a instalação de toda infraestrutura necessária, os equipamentos que compunham o BDA Fase I foram transferidos para o compus do INPE em Cachoeira Paulista (longitude:$45^o0'20"$ W e latitude: $22^o41'19"$). A máxima linha de base foi aumentada para uma distância de 216 metros para testes do sistema receptor analógico. A configuração geo-

métrica das antenas foi escolhida de forma a se obter uma cobertura mais uniforme possível no plano *uv* (Figura 2.42) (FARIA, 2006).

Um correlacionador digital de 32 canais foi desenvolvido pelo *Indian Institute of Astrophysics* (IIA/Índia) para digitalização e correlação dos sinais provenientes dos receptores das cinco antenas do BDA Fase I. A obtenção das primeiras franjas solares e não solares utilizando o BDA Fase I permitiram testar os módulos dos sistema do BDA, incluindo o sistema de ajuste de atraso digital, correlacionador e o sistema de aquisição, tratamento e visualização dos dados.

Figura 2.42: Foto das 5 antenas instaladas no site de Cachoeira Paulista.

A Fase II teve início em 2006 e foi concluída em 2012, onde o arranjo foi formado por 26 antenas em forma de um "T", onde 17 antenas formam uma linha Leste-Oeste e 9 antenas na direção Sul, operando em uma faixa de frequência de 1.2 a 1.7 GHz, onde a menor linha de base é de 9m e a maior é de 252m, o que resulta em uma resolução espacial de aproximadamente 2.9x3.4 arc min 1.4 GHz, na Tabela 2.7 é apresentado a relação de linhas de base do arranjo do BDA concluída na segunda fase (SUBRAMANIAN, 2012).

Multiplic	baseline	multiplic	Baseline	multiplic	baseline	multiplic	baseline	multiplic	baseline
e1xe4	72	e4xe8	45	e8xe10	18	e10xe13	27	e13xe16	54
e1xe8	117	e4xe10	63	e8xe13	45	e10xe16	81	e13xs1	37.11
e1xe10	135	e4xe13	90	e8xe16	99	e10xs1	12.73	e13xs4	50.91
e1xe13	162	e4xe16	144	e8xs1	12.73	e10xs4	37.11	e13xs7	96.93
e1xe16	216	e4xs1	54.75	e8xs4	37.11	e10xs7	90.45	e13xs9	165.95
e1xs1	126.32	e4xs4	64.9	e8xs7	90.45	e10xs9	162.25		
e1xs4	128.86	e4xs7	104.96	e8xs9	162.25				
e1xs7	154.84	e4xs9	170.76						
e1xs9	205.23								

Multiplic	baseline	multiplic	baseline	multiplic	baseline	multiplic	Baseline
e16xs1	90.45	s1xs4	27	s4xs7	54	s7xs9	72
e16xs4	96.93	s1xs7	81	s4xs9	126		
e16xs17	122.28	s1xs9	153				
e16xs9	185.32						

Tabela 2.7: Linhas de Base do BDA: Multiplic = antenas que formam a linha de base e baseline = comprimento, em metros, da linha de base.

Cinco antenas na direção East-West começaram a ser utilizadas para observação solar em Setembro de 2011, e em 2012 esse número passou para 10, em seguida 21 até completar o número de 26 antenas, meta da segunda fase da construção do arranho, junto com a instalação dos subsistemas de *front end* de RF, o oscilador local e os recerivers de IF, que também foram instalados e testados. A Figura 2.43 mostra o arranjo de testes do BDA com 10 antenas e a Figura 2.44 o resultado da visibilidade no plano *uv* correspondente (SUBRAMANIAN, 2012).

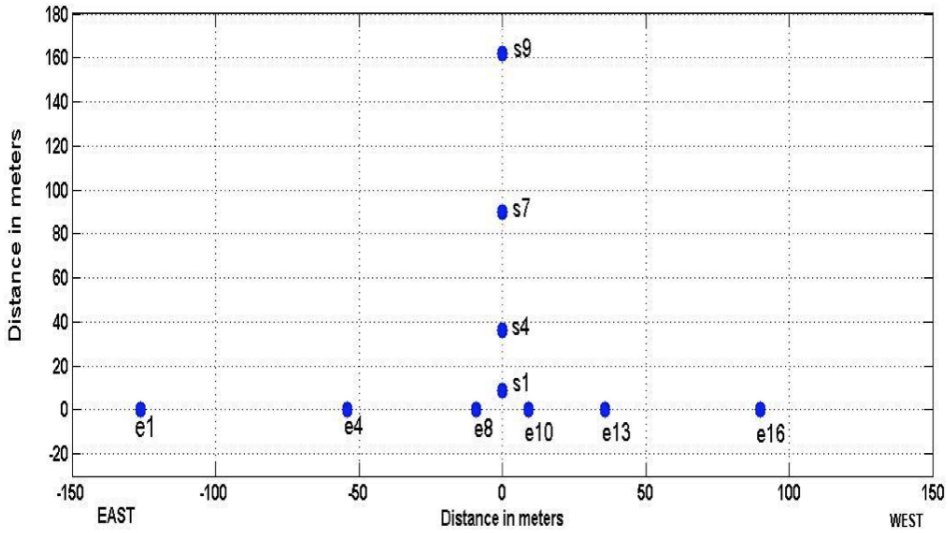


Figura 2.43: Arranjo de teste do BDA utilizando 10 antenas(SUBRAMANIAN, 2012).

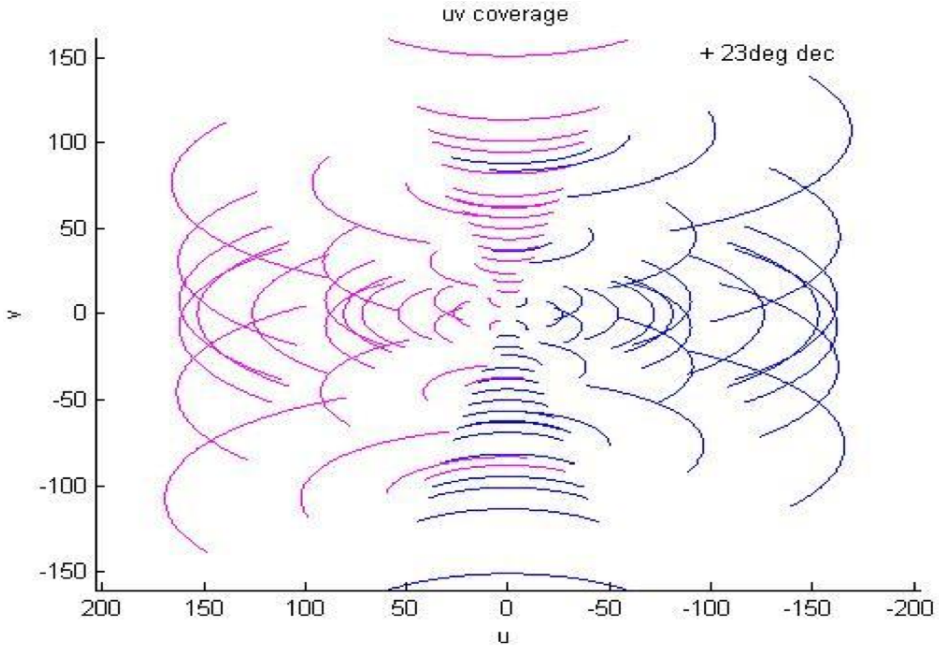


Figura 2.44: Amostragem do plano *uv* do arranjo de 10 antenas (SUBRAMANIAN, 2012).

Capítulo 3

DESCRIÇÃO DA PLATAFORMA COP-WS E METODOLOGIA

No capítulo anterior foi apresentada a base teórica utilizada para o desenvolvimento deste trabalho de doutorado, que tem por objetivo a implementação de uma plataforma computacional de alto desempenho que apresente a capacidade de processamento de grandes volumes de dados em pipeline e de forma contínua. Neste capítulo é apresentada a metodologia de trabalho adotada, no desenvolvimento desta plataforma, estando organizado em duas partes: a primeira com as seções que tratam do desenvolvimento da plataforma em si; e a segunda parte que apresenta o uso desta plataforma no desenvolvimento de duas aplicações que serão utilizadas como provas de viabilidade, ou validação, quanto ao uso da plataforma implementada.

Uma das questões descritas por Mars e seus colaboradores (MARS; TANG; HUNDT, 2011), e apresentada na Figura 1.1, é a diferença do desempenho de uma mesma aplicação em diferentes arquiteturas. Este trabalho de tese comprova a existência de necessidades especiais, e particulares, de cada tipo/lógica das rotinas computacionais, e suas exigências na utilização da plataforma/arquitetura escolhida. Um dos conceitos importantes da tecnologia de WS é a sua capacidade de permitir interoperabilidade entre ambientes computacionais, e é justamente esta característica, que faz com que esta tecnologia permita a criação de um ambiente heterogêneo, onde diferentes tecnologias/arquiteturas possam ser integradas de forma a criar uma plataforma colaborativa, onde cada rotina, de uma determinada aplicação, possa ser implementada dentro de um ambiente que melhor atenda as suas necessidades.

E é este o objetivo da plataforma de Coprocessamento Paralelo Distribuído por *Web Services* CoP-WS, implementada neste trabalho. A criação de um ambiente híbrido de tecnologias, que permita a utilização da tecnologia que melhor se adequa ao processamento de uma deter-

minada aplicação, utilizando um ambiente distribuído por WS, e que permita o processamento de dados de uma forma contínua (fluxo de dados), numa feição *pipeline*.

3.1 Conceitos da Plataforma CoP-WS

Conforme introdução na Seção 1.2, a plataforma CoP-WS consta das seguintes tecnologias: 1) WS; 2) XML; 3) GPU. A ideia básica da plataforma CoP-WS é a utilização de um conjunto de máquinas, onde cada qual faz o papel de provedora e requerente de serviços, sendo que estes serviços são funcionalidades, e seu conjunto representa uma aplicação completa. Cada máquina é composta por um par CPU-Coprocessador, e possui uma pilha de software, que é responsável pela execução dos serviços e gerência do fluxo de processamento dentro da plataforma, usando a tecnologia de WS como interface de comunicação máquina-a-máquina. Por coprocessador pode ser entendido qualquer dispositivo que apresente um perfil mais adequado para o processamento de uma determinada funcionalidade, como por exemplo FPGA's e GPU's.

Desta forma, cada máquina pode ser vista como um **nó computacional**, especializado no processamento de uma determinada rotina, sendo capaz de atender suas particularidades, de processamento, e o WS passa a funcionar como um mecanismo de abstração deste nó computacional, encapsulando sua implementação, e disponibilizando uma API de comunicação, por meio da chamada de seus serviços, conceito muito semelhante ao encontrado no paradigma de programação orientada a objetos.

Na Figura 3.1 podem ser vistos os componentes que formam um nó computacional, dentro da plataforma CoP-WS. Como pode ser observado, o CPU é o elemento base da plataforma, e todos os demais elementos são acoplados de forma a estender ou especializar seu processamento, que na Figura 3.1 está sendo representado pelos Dispositivos de Apoio ao Processamento, que funcionam como um Coprocessador. Dentro de cada nó computacional existe um conjunto de rotinas que formam uma pilha de software, sendo que cada camada é responsável por uma parte do processamento, seja da execução do serviço, ou de rotinas auxiliares, necessárias ao controle do fluxo de processamento da aplicação e/ou plataforma como um todo, além do próprio WS que funciona como uma API de comunicação/acesso a este nó computacional.

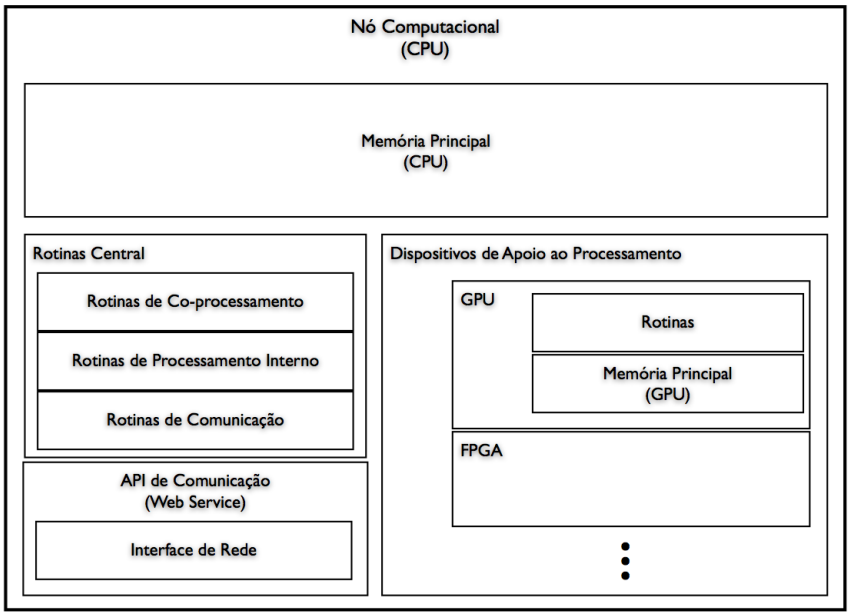


Figura 3.1: Representação dos componentes de um nó Computacional da Plataforma CoP-WS.

Considerando-se que uma determinada aplicação possa ser dividida em um conjunto de rotinas, e cada nó computacional seja especializado na execução de uma única rotina, para se ter o processamento de uma aplicação completa, é necessário que sejam utilizados mais de um nó computacional. O conjunto desses nós é chamado de **arranjo computacional**, e cada um de seus elementos - nós computacionais - são especializados na execução de cada uma das rotinas que compõe a aplicação que será processada dentro da plataforma CoP-WS.

Uma característica interessante da Plataforma CoP-WS é a alto suficiência que cada um dos nós computacionais tem de sua existência dentro do fluxo de processamento de uma determinada aplicação, ou seja, uma vez que o arranjo computacional esteja configurado, cada nó computacional, sem a intervenção humana, sabe o que fazer com o resultado de seu processamento, seja enviar este resultado para o nó computacional responsável pelo processamento do próximo estágio, ou a finalização de um ciclo de processamento por meio de execução de rotinas especializadas, como o acesso a um banco de dados, ou ainda geração de imagens e/ou arquivos em disco.

É esta alto suficiência de sua participação dentro da lógica da aplicação, associada ao fato de que cada nó computacional processa apenas uma rotina dentro do fluxo de processamento, que permite que a execução de uma aplicação dentro da plataforma CoP-WS seja realizada em pipeline, ou seja, cada nó computacional processa uma massa de dados adquirida em um tempo diferente segundo os ciclos de processamento.

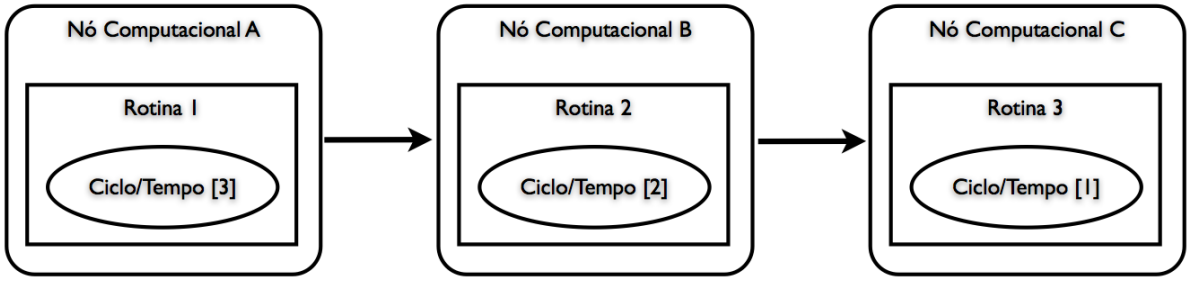


Figura 3.2: Exemplo de um arranjo computacional formado por três objetos.

Na Figura 3.2 é apresentado um exemplo de um arranjo computacional, formado por três nós computacionais chamados de A, B e C, responsáveis pelo processamento das rotinas 1, 2 e 3 respectivamente. Uma vez que cada nó computacional conclua a execução de sua rotina, envia o resultado de seu processamento para o próximo nó computacional, responsável pelo processamento da rotina do próximo estágio, e para não ficar ocioso, recebe a massa de dados do próximo ciclo de processamento. É por esta feição, que cada um dos nós computacionais A, B e C, estão processando dados de ciclos diferentes.

Existem dois problemas inerentes a qualquer arquitetura de computação de alto desempenho, que são: volume de dados a ser processado e tempo de resposta desejado. Para flexibilizar a plataforma CoP-WS, de forma a atender estes dois problemas, foi criado o conceito de **bloco funcional**. Um bloco funcional é o conjunto de nós computacionais, configurados dentro do mesmo arranjo computacional, e que oferecem o mesmo serviço. Por meio da utilização de um bloco funcional, é possível realizar o processamento paralelo da massa de dados de um determinado estágio de processamento. A necessidade de sua configuração se dá, de forma a priori, em dois cenários: 1) limitação do nó computacional, caso em que a massa de dados excede a capacidade de memória do nó computacional, seja do CPU ou mesmo do Coprocessador utilizado; e 2) tempo de resposta esperado, caso em que, mesmo a massa de dados não excedendo a capacidade do nó computacional, o tempo gasto para seu processamento é maior do que o esperado.

Na Figura 3.3(a) é apresentado um arranjo computacional formado por cinco nós computacionais, para o processamento de uma aplicação que possui três funcionalidades (A, B e C), onde pode ser observado um bloco funcional com três nós computacionais, para o processamento da Funcionalidade B. Um caso especial na configuração de um bloco funcional, é quando se utiliza um nó computacional como porta de acesso para um outro arranjo computacional, um cluster por exemplo, sem a necessidade de que este segundo arranjo seja da plataforma CoP-WS, como pode ser observado na Figura 3.3(b).

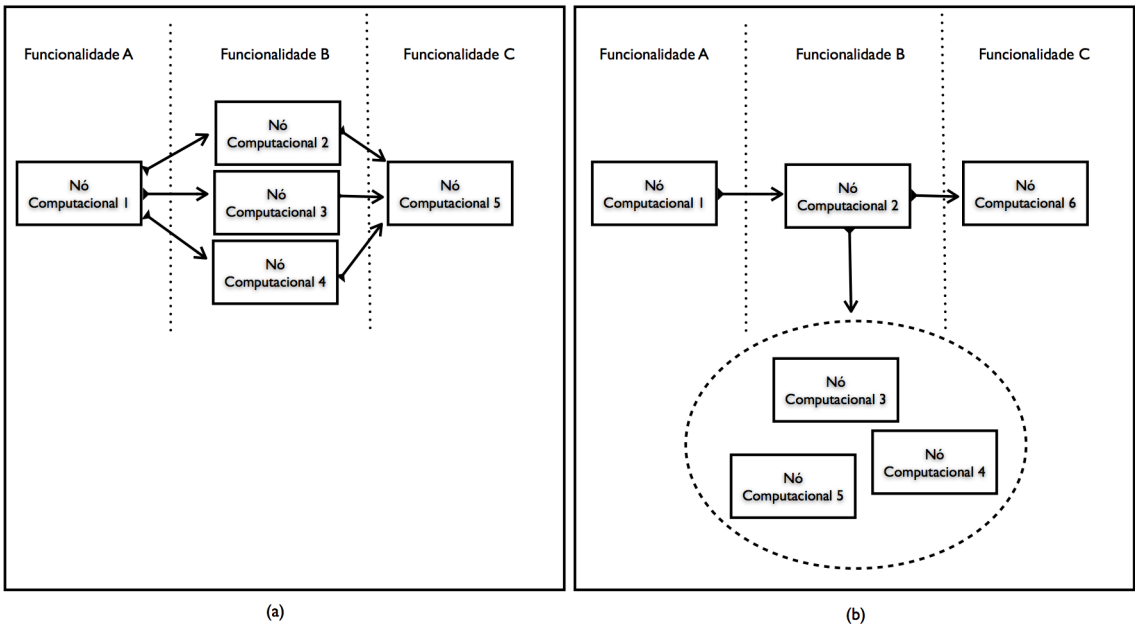


Figura 3.3: Exemplo de arranjos computacionais configurados com a utilização de blocos funcionais.

A utilização de blocos funcionais faz com que seja necessária a utilização de mecanismos para o tratamento de dados segmentados. Uma vez que, como pode ser visto na Figura 3.3(a), o nó computacional 1, deve ser capaz de segmentar as massa de dados resultante de seu processamento em três blocos, onde cada um destes blocos de dados serão enviados para os nós computacionais, 2, 3 e 4. Por sua vez o nó computacional 5 deve ser capaz se processar uma massa de dados composta pelo resultado do processamentos dos objetos computacionais 2, 3 e 4. De forma a atender as necessidades deste problema, a plataforma CoP-WS utiliza o conceito de **envelope de dados**, onde os dados são organizados, segundo uma chave de classificação, individual, podendo haver redundância dos dados para facilitar a segmentação e permitir o processamento individual e independente de cada um dos nós computacionais do arranjo.

3.2 Implementação da Plataforma CoP-WS

Todo o processo de implementação da plataforma CoP-WS teve como objetivo abstrair a complexidade do processo de comunicação dos dados dentro do arranjo computacional, de forma a permitir que o desenvolvedor da aplicação pudesse se dedicar na implementação das funcionaliades do processo a ser executado em um nó computacional, o resultado deste processo pode ser visto na Figura 3.4, onde pode é apresentado dois blocos de código, a organização das classes do CoP-WS, a esquerda organizadas nas classes Service, Message e Package e a classe a direita, responsável pelo processamento da Transformada de Hilbert e que utiliza (extende) as

funcionalidades da classe Service.

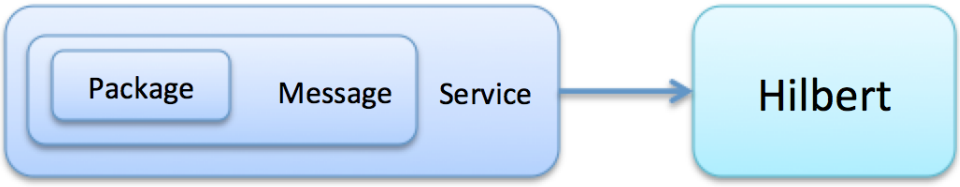


Figura 3.4: Organização da Implementação da aplicação dentro da plataforma CoP-WS.

Como cada nó computacional é responsável pelo processamento de um único processo descrito dentro do fluxo de processamento do problema, cada nó é formado pela biblioteca CoP-WS e pelo código da rotina a ser executada pelo nó. Sua implementação segue o paradigma da OO (Orientação a Objetos) o que facilita a questão de reutilização do código por meio da herança entre as classes. Desta forma, a classe que irá ser responsável pelo processamento da rotina no nó computacional, deve extender a classe Service da biblioteca do CoP-WS (Figura 3.5).

```
1  using CoP-WS::Service;
2
3  class Service {
4    private:
5      String        srv_name;
6      int           size_b_in;
7      int           size_b_out;
8      int           qtd_n_in;
9      int           qtd_n_out;
10     msg_queue*    queue_in;
11     msg_queue*    queue_out;
12     node_address* queue_back;
13     node_address* queue_forward;
14     bool          log;
15
16   public:
17     Service(String srv_name, int size_b_in, int size_b_out,
18                              int qtd_n_in,  int qtd_n_out);
19     ~Service();
20
21     int  getSize();
22     int  setSize(int size);
23     int  addMessageIn(Message *msg);
24     int  addMessageOut(Message *msg);
25     int  validateIn(int time_ref);
26     int  validateOut(int time_ref);
27 };
28 _
```

Figura 3.5: Código C++ da classe Service da biblioteca CoP-WS.

A classe Service conta com duas configurações importantes que são: a definição do nó dentro do fluxo de comunicação e o tamanho do buffer de processamento (Figura 3.6). A definição do nó dentro do fluxo de comunicação esta relacionada com a determinação do número de nós que se encontram no bloco funcional que antecede (*qtd_n_in*) e no seguinte (*qtd_n_out*) ao seu

processamento. Estas duas informações são utilizadas no processo de validação dos pacotes recebidos, uma vez que uma massa de dados só pode ser considerada em condições de processamento pelo nó atual se todos os nós anteriores enviarem o resultado de seu processamento de um mesmo ciclo. E o resultado de seu processamento pode ser fragmentado ou dividido para o número de nós do estágio seguinte.

Cada nó conta ainda com duas filas que funcionam como um buffer para auxiliar em casos de possíveis atrasos da rede, sendo estas filas de entrada de dados (*queue_back*) e no envio dos dados resultantes do processamento (*queue_forward*). A fila de entrada é responsável por organizar os pacotes recebidos pelos nós do estágio anterior ao nó atual. O pacote de comunicação utilizado na transferência dos dados entre os nós conta com uma chave formada por três atributos:

- Número do ciclo: gerado pelo primeiro estágio do fluxo de processamento, na fase de aquisição dos dados e que irá acompanhar os dados durante todo o ciclo de processamento, mesmo em casos em que os dados serão distribuídos e reagrupados, estes nunca perderam seu número de geração;
- Número total de pacotes: quando um determinado nó divide a massa de dados resultante de seu processamento para envio ao bloco funcional seguinte, para cada nós do próximo estágio é informado o número total dessa divisão. Este número segue para o próximo estágio e é utilizado para número de validação para ter certeza que todos os pacotes, de um determinado ciclo de processamento, foram reagrupados; e o
- Número do pacote atual: que informa qual é o número de indice do pacote atual que o nó computacional esta recebendo.

Desta forma as filas, de entrada e saída permitem "ocultar" o tempo de espera que pode ocorrer em alguns casos tanto para a conclusão do processamento de dados e seu envio quanto para o recebimento de uma massa de dados fragmentada que necessita esperar por algum pacote para então dar inicio ao processamento do ciclo determinado.

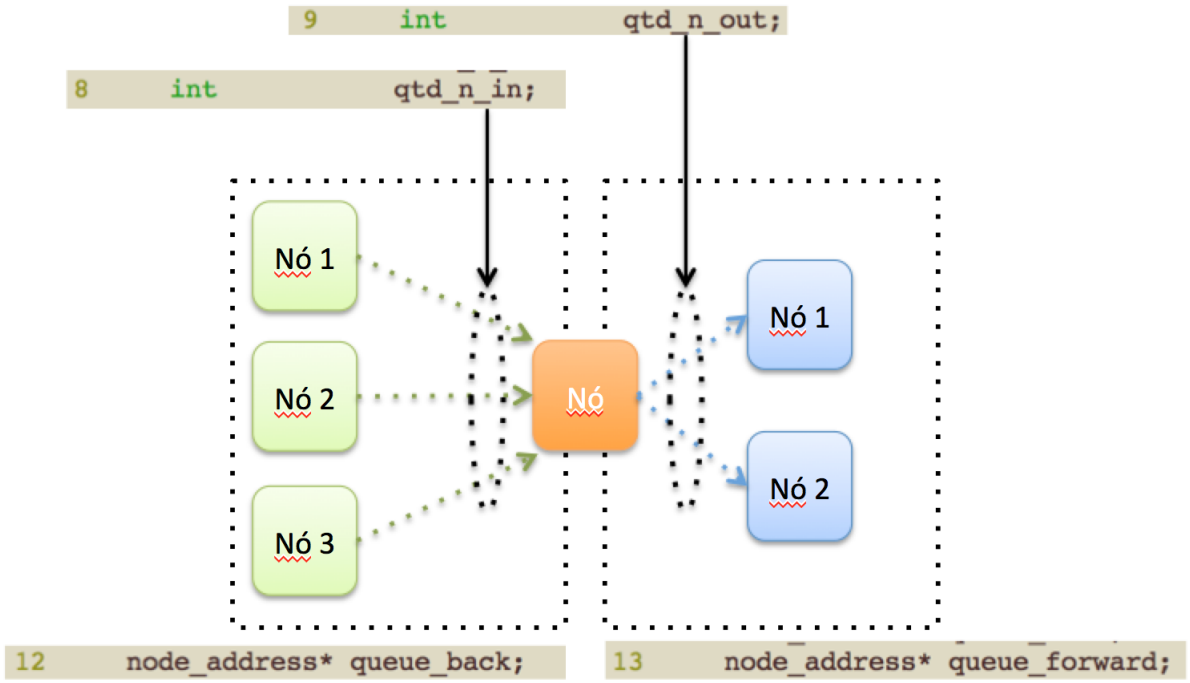


Figura 3.6: Fluxo de Comunicação entre os Nós Computacionais.

Um objeto da classe Service é formado por um par de objetos da classe Message (Figura 3.7), que funciona como repositório para a massa de dados de entrada (*queue_in*) e de saída (*queue_out*), onde o tamanho destas filas é determinado pelos atributos específicos.

```
1  using CoP-WS::Message;
2
3  class Message{
4    private:
5      int tot;
6      Package *packages;
7    public:
8      Package(int total);
9      Package(int total, Package *packages);
10     ~Package();
11
12     int      getTotal();
13     void     setTotal(int total);
14     int      addPackage(Package pkg);
15     int      setPackages(Package *pkg);
16     int      validate();
17     Package* getPackages();
18     Package* getPackage(int idx);
19 };
20
```

Figura 3.7: Código C++ com a especificação da Classe Message.

A base da plataforma CoP-WS consiste na capacidade da troca de mensagens entre os nós computacionais de diferentes blocos funcionais seguindo o fluxo de processamento dos dados, coletados em diferentes tempos, ou seja, em *pipeline*. Para que esta capacidade seja atendida o *kernel* da plataforma CoP-WS conta com uma classe chamada *Package* (Figura 3.9)

que funciona como um container identificado para um vetor de dados do tipo *void*. O uso de ponteiros do tipo *void*, se deve ao fato de que a plataforma CoP-WS não sabe qual é a estrutura de dados que estará sendo utilizada para processamento em cada um dos nós e este recurso permite que uma massa de dados possa ser transferida entre os nós do arranjo, para que então, no nó destino, passe por um processo de *cast* especializando o ponteiro para o tipo/estrutura de dados específica do bloco funcional.

A Classe Message é utilizada como elemento de particionamento da massa de dados, sendo utilizada para o envio dos dados durante o processo de comunicação entre os nós computacionais participando do arranjo. Esta classe possui três atributos especiais que são utilizados para identificar uma massa de dados, armazenada no atributo **data_message**, que são: 1) **index**, responsável pela identificação de uma das partes de divisão dos dados; 2) **total**, indica o número máximo de segmentos que são realizados na massa de dados; e 3) **reference**, responsável por identificar o tempo, ou ciclo de processamento, em que esses dados são segmentados. Com base nestes três dados é possível ter a informação de que um determinado nós possui os dados (atributo data_message), referentes à segmentação 2 (atributo index), de um total de 5 segmentos (atributo total), que foi processado no ciclo de número 23 (atributo reference).

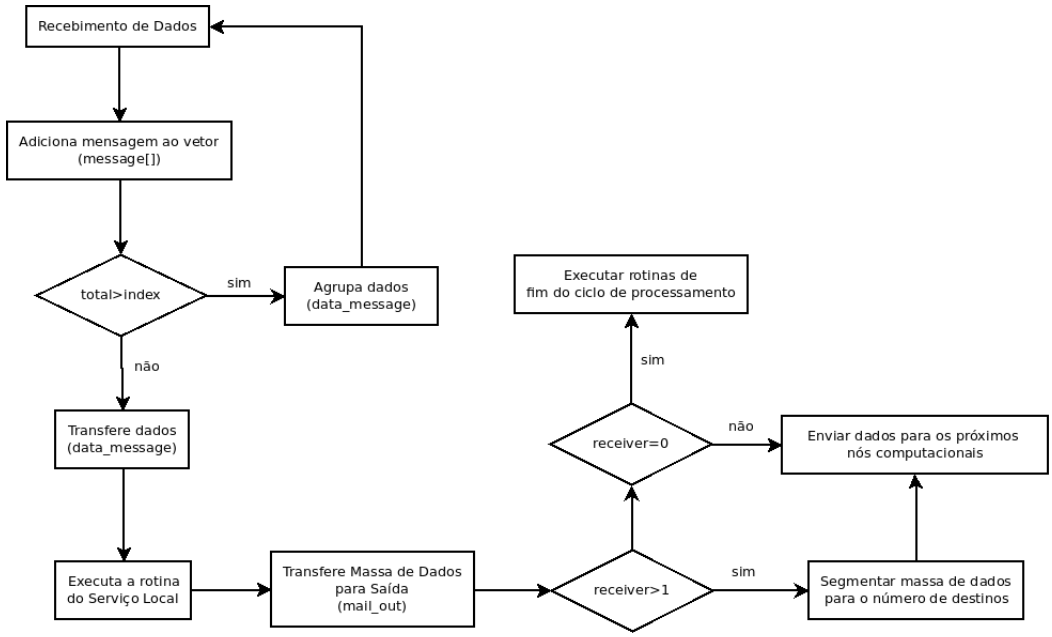


Figura 3.8: Fluxograma da lógica do tratamento de dados da plataforma CoP-WS.

Na Figura 3.8 é apresentado um fluxograma do processo de recepção dos envelopes de dados pelas classes da package copws. O processo tem início com o *Recebimento de Dados*, que ocorre no momento da chamada do serviço a ser executado pelo nó computacional. Com o envelope de dados recebido a classe Service, trabalhando com a Classe Mail, por meio do

atributo (objeto) mail_in, cria um objeto da classe Message e *Adiciona mensagem ao vetor*, message do objeto mail_in. Neste ponto é verificado se o número total, um dos atributos da classe Message, é maior que 1, o que indica o número de nós computacionais do bloco funcional que antecede o nó computacional. Se isto for verdadeiro, total > 1, é necessário aguardar o recebimento de dados os envelopes de dados, e durante seu recebimento realizar o *Agrupamento dos dados* destes envelopes no atributo data_message do objeto mail_in.

A execução da rotina do serviço a ser executado pelo nó computacional, utiliza como massa de dados de entrada o atributo data_message do objeto mail_in. No final da execução da rotina do serviço, é verificada a quantidade de endereços existentes no atributo receiver da classe Service. Este número indica o próximo estágio do fluxo de processamento. Caso não existam endereços para os quais devam ser enviados a massa de dados resultante, esta deve ser processada localmente por meio do uso de um API especialida, como o acesso a banco de dados, acesso a disco, etc. Caso contrário, receiver > 0 a massa de dados deve ser enviado para o próximo bloco funcional, havendo dois casos possíveis: 1) existe apenas um nó computacional para o próximo estágio,. caso em que a massa de dados é colocada em um envelope de dados tendo o atributo index com o valor 1 e o atributo total 1; 2) a massa de dados deve ser enviada a mais de um nó computacional, caso em que a massa de dados é segmentada em diferentes envelopes de dados, com o valor do atributo index referente a cada segmento, e o atributo total referente ao número de receiver.

Cada objeto *Package* representa uma amostra que é agrupada em uma conjunto de dados representada pela Classe *Message* do CoP-WS (Figura 3.7), desta forma um objeto *Message* representa o *dataset* a ser processador em cada ciclo de processamento do arranjo, como pode ser observado na Figura 3.10.

```
1  using CoP-WS::Package;
2
3  class Package{
4    private:
5      int idx;
6      void* data;
7    public:
8      Package(int idx, void *data);
9      ~Package();
10
11     int    getIndex();
12     void  *getData();
13     int    setData(void *data);
14 };
```

Figura 3.9: Código C++ com a especificação da Classe Package.

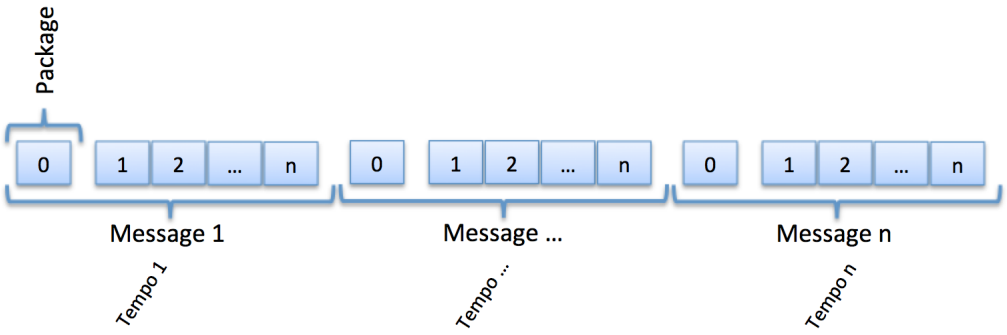


Figura 3.10: Representação em blocos do relacionamento dos objetos da Classe Package na formação do objeto Message.

Neste trabalho cada um dos nós computacionais foi configurado para realizar o processamento de uma única rotina. Assumindo por exemplo a aplicação do correlacionador de sinais, foram implementados quatro processos: as transformadas de Hilbert e a FFT, a multiplicação cruzada das linhas de base e a integração dos resultados da multiplicação. A versão atual da plataforma foi implementada em C++ e desta forma como ferramenta para o desenvolvimento do WS de cada um destes nós foi utilizado o gsoap (EGELEN, 2013). As assinanturas de cada um dos serviços implementados Hilbert, FFT, Multiplicação e Integração, pode ser visto nas Figuras 3.11, 3.12, 3.13 e 3.14 respectivamente, cuja implementação segue a sintaxe proposta pela biblioteca gsoap, onde:

- Linha 01: indica o nome do arquivo header;
- Linha 02: refere-se o nome do serviço que esta sendo implementado;
- Linha 03: *namespace* do código;
- Linha 04: endereço do WS e
- Linha 05: assinatura do método a ser chamado sendo os parâmetros:
  - *time_cyle*: time referente ao ciclo de integração utilizado como controle pelo ciclo de pipeline;
  - *total*: número total de pacotes de dados desta estrutura de dados;
  - *index*: valor de referência do pacote que este sendo passado a um determinado nó computacional; e
  - *signal_array, ds_hilbert, ds_base_line e ds_mult*: ponteiros para a estrutura de dados a ser processada pelo serviço, transformada de hilbert, FFT, multiplicação cruzada e integração respectivamente.

Os arquivos XML que descrevem o protocolo WSDL de cada um destes, são apresentados no Apêndice A.

```
1 //File: nodeHilbert.h
2 //gsoap ns service name: nodeHilbert
3 //gsoap ns service namespace: urn:nodeHilbert
4 //gsoap ns service location: 200.18.98.156/nodeHilbert.cgi
5 int ns__nodeHilbert(time_t& time_cycle, int& total, int& index, void& signal_array);
```

Figura 3.11: Código C++ da assinatura da chamada da transformada de Hilbert.

```
1 //File: nodeFFT.h
2 //gsoap ns service name: nodeFFT
3 //gsoap ns service namespace: urn:nodeFFT
4 //gsoap ns service location: 200.18.98.67/nodeFFT.cgi
5 int ns__nodeFFT(time_t& time_cycle, int& total, int& index, void& ds_hilbert);
```

Figura 3.12: Código C++ da assinatura da chamada da transformada de Fourier.

```
1 //File: nodeMultiplication.h
2 //gsoap ns service name: nodeMultiplication
3 //gsoap ns service namespace: urn:nodeMultiplication
4 //gsoap ns service location: 200.18.98.154/nodeMultiplication.cgi
5 int ns__nodeMultiplication(time_t& time_cycle, int& total, int& index, void& ds_base_line);
```

Figura 3.13: Código C++ da assinatura da chamada da Multiplicação Cruzada.

3.3 Prova de Viabilidade

Para realizar a prova de viabilidade da plataforma CoP-WS, é proposta a implementação de dois aplicativos, ambos dentro da área de radiointerferometria para observações solares, sendo estes:

- Correlacionador de sinais complexos para um arranjo interferométrico; e
- Sistema para o reconhecimento de explosões solares.

Na Figura 3.15 é apresentado o cenário sobre o qual as duas aplicações foram projetadas, onde pode ser observado que o sinal é proveniente de um arranjo de antenas, que passa por um correlacionador de sinais complexo de arquitetura FX, primeira aplicação de implementada. O produto do processo de correlação dos sinais das antenas do arranjo, passa por um processo de deconvolução, o que gera a imagem do evento observado. Esta imagem pode então ser processada, com o objetivo de realizar o reconhecimento das explosões solares, segunda aplicação implementada.

```
1 //File: nodeIntegration.h
2 //gsoap ns service name: nodeIntegration
3 //gsoap ns service namespace: urn:nodeIntegration
4 //gsoap ns service location: 200.18.98.151/nodeIntegration.cgi
5 int ns__nodeIntegration(time_t& time_cycle, int& total, int& index, void& ds_mult);
```

Figura 3.14: Código C++ da assinatura da chamada da Multiplicação Cruzada.

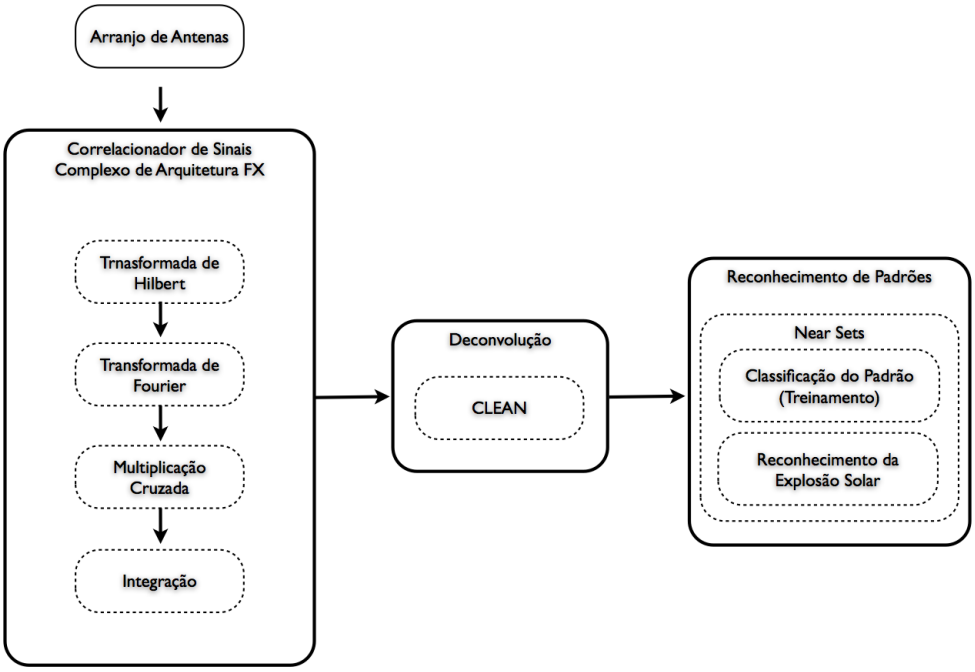


Figura 3.15: Fluxo de processamento para a obtenção da imagem do radio interferômetro.

A configuração do arranjo interferométrico adotado por este trabalho, foi inspirado no BDA, descrito na Seção 2.16, contudo, este instrumento não possui uma interface que permita que os dados adquiridos pelas antenas, possam ser processados pela atual versão desta plataforma, de forma que foram implementadas pequenas aplicações que simulam o volume de dados a ser processado segundo a configuração mantida pelo BDA.

3.3.1 Correlacionador de sinais complexos

Um correlacionador de sinais pode ser definido como sendo um dispositivo responsável pelo processamento das amostras do plano espectral *uv* da visibilidade complexa obtida pela observação do objeto celeste pelas antenas de um arranjo, detalhes sobre seu funcionamento podem ser encontradas na Seção 2.15, e um diagrama de blocos com um resumo do correlacionador complexo com uma arquitetura FX pode ser vista na metade esquerda da Figura 3.15.

A exigência por um tempo de processamento "rápido", em muitos casos chegando a um processamento em tempo real, faz com que a escolha natural para o desenvolvimento deste

dispositivo seja uma implementação em *hardware*, onde, no caso do BDA, foi realizada uma implementação usando FPGA, cujos detalhes são descritos no relatório técnico, enviado a FAPESP, apresentado por Subramanian (SUBRAMANIAN, 2012).

A pesquisa por uma implementação em software dos correlacionadores, é atraente, concentrando-se muitas vezes no processo de correlação de sinais de instrumentos como o VLBI (Very-long-baseline interferometry) (ALTAMIMI; COLLILIEUX; MÉTIVIER, 2011; Rygl, K. L. J. et al., 2012), onde não existe a possibilidade de processamento em tempo real, devido a questões de configuração dos próprios instrumentos. Contudo, com a evolução de dispositivos como o GPU e o processador Cell da IBM, a possibilidade de implementação de um sistema de correlação de sinais em tempo real, vem se tornando uma realidade próxima, como pode ser observado em trabalhos como o Harris(HARRIS, 2009) e de Woods (WOODS, 2010).

A escolha da implementação de um correlacionador de sinais complexo de arquitetura FX, como um instrumento de teste de viabilidade da plataforma CoP-WS, se deve as seguintes características deste tipo aplicação:

1. Fluxo contínuo na produção dos dados: o correlacionador possui como entrada de dados os sinais amostrados por um arranjo de antenas que observam um mesmo ponto na esfera celeste. Cada instrumento, possui um tempo de integração, que indica o tempo de duração de um ciclo de seu processamento, correspondendo a obtenção de dados suficientes para a construção de uma imagem do objeto celeste. Isso indica que os processamentos a serem executados pelo correlacionador são realizados repetidas vezes, o quanto durar o tempo de observação do instrumento;

2. Tempo de resposta curto: um outro fator importante de um correlacionador, é que este deve realizar o processamento da correlação dentro de um período de integração do instrumento. Isso quer dizer que, no caso do BDA, os processos de Transformada de Hilbert, FFT, Multiplicação Cruzada e integração dos resultados, devem ser realizados em um período daproximadamente 100 ms para se obter imagens em tempo próximo ao real ; e

3. Volume de dados é significativo: o volume de dados produzido pelas antenas de um arranjo interferométrico é significativo, como apresentado na Tabela 2.6. No caso do BDA a cada ciclo de integração são processados aproximadamente 5 GB de dados.

A primeira fase do desenvolvimento da aplicação deve ser uma análise sobre o fluxo de dados e processamento do problema que se estuda, neste caso do correlacionador, são encontrados os quatro processos a serem implementados: 1) transformada de Hilbert; 2) transformada de

Fourier (FFT); 3) Multiplicação cruzada; e a 4) Integração. A Figura 3.16 apresenta o fluxograma da aplicação do correlacionador implementada neste trabalho, onde os blocos funcionais destacados correspondem esses quatro processos.

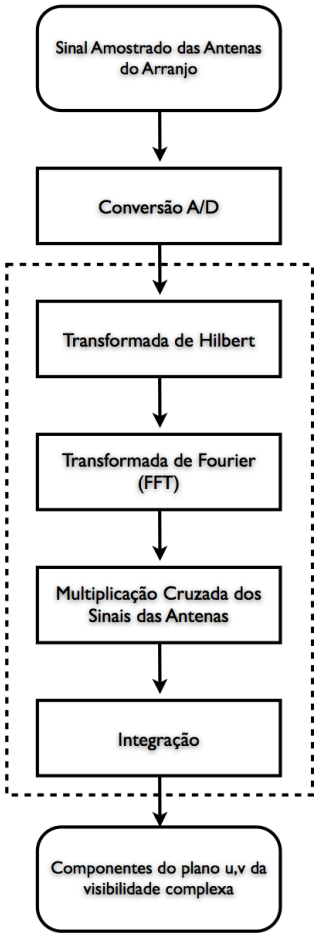


Figura 3.16: Fluxograma da aplicação do correlacionador FX implementado.

Como descrito, a massa de dados de entrada na aplicação de correlação de sinais, são os dados amostrados pelas antenas do arranjo. Apesar da implementação deste correlacionador ser inspirado no BDA, não foram utilizados dados reais para seu processamento, isso porque o instrumento real não possui uma interface de entrada para o CoP-WS. A ausência desta interface, ou de dados que pudessem ser utilizados como entrada para a aplicação exigiu que os dados amostrados das antenas fossem simulados.

O simulador de sinais amostrados que foi implementado neste trabalho, leva em consideração apenas o número de antenas de uma arranjo, não havendo uma preocupação formal com a coerência do sinal produzido. O objetivo, neste momento é o de gerar uma massa de dados suficientemente grande que fosse representativa, segundo a configuração do BDA.

O simulador implementado produz uma matriz, onde o número de linhas refere-se ao nú-

mero de antenas do arranjo, e o número de colunas ao tamanho do vetor de amostras utilizado. Em cada posição desta matriz, existe um valor real (ponto flutuante) que representa o valor cosseno observado por cada uma das antenas em um determinado período de tempo. Esta matriz é utilizada como entrada para o primeiro bloco funcional do correlacionador, a Transformada de Hilbert, Figura 3.17, matriz à esquerda.

A Transformada de Hilbert é responsável pela obtenção do valor de seno do sinal amostrado. De posse desses dois valores, seno e cosseno, do sinal "amostrado", a Transformada de Hilbert produz então uma segunda matriz, seguindo o padrão da primeira (linhas = número de antenas, e colunas = ao tamanho do vetor de amostras), contudo o valor armazenado em cada posição passa a ser um número complexo, onde a parte imaginária é formada pelo seno e a parte real é formada pelo cosseno, Figura 3.17, matriz à direita.

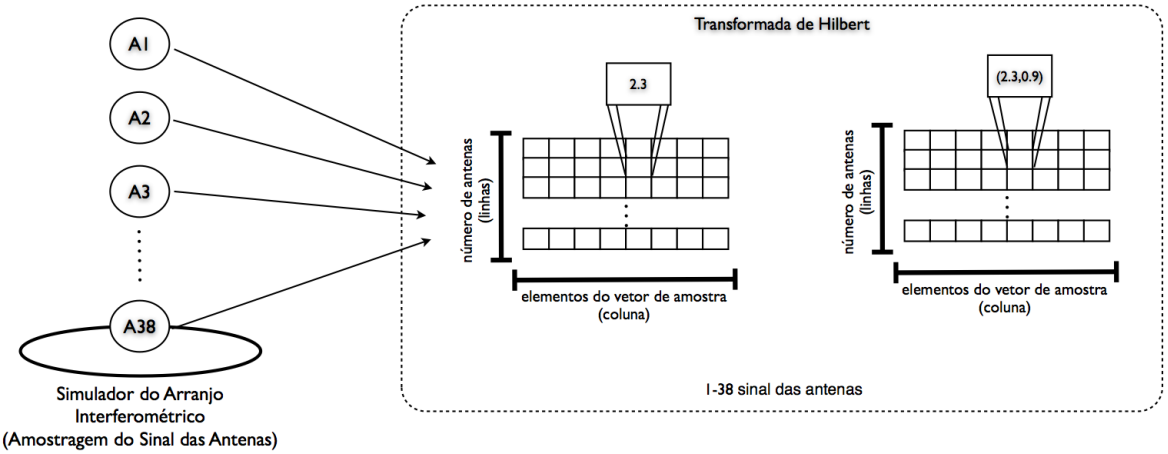


Figura 3.17: Representação da estrutura de dados de entrada e saída do bloco funcional da Transformada de Hilbert.

O segundo bloco funcional é a Transformada de Fourier, que tem as estruturas de dados de entrada e saída apresentadas na Figura 3.18. Este bloco funcional recebe uma matriz resultante do processamento do bloco anterior (Transformada de Hilbert) onde o número de linhas refere-se ao número de antenas e o número de colunas é o número de elementos amostrados pela antena representados por um número complexo, Figura 3.18, matriz à esquerda. A transformada de Fourier desta matriz resulta em uma outra matriz, de mesmas dimensões, contudo com seus valores no domínio da frequência, matriz à direita.

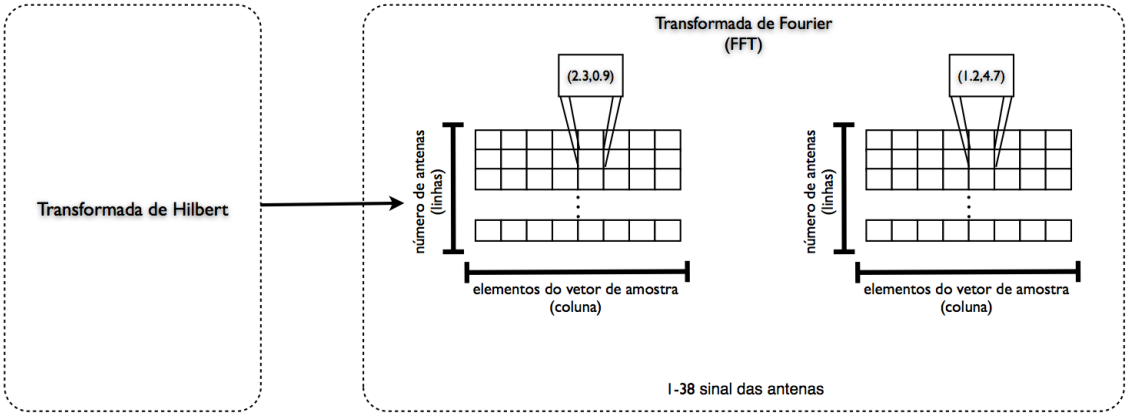


Figura 3.18: Representação da estrutura de dados de entrada e saída do bloco funcional da Transformada de Fourier (FFT).

Todas as estruturas de dados utilizada na aplicação do correlacionador seguem o escopo da filosofia proposta para a implementação de uma aplicação dentro da plataforma CoP-WS, onde os dados são organizados de forma que possam ser segmentados e processados de forma independente, dentro dos nós computacionais que formam um arranjo computacional, o que leva muitas vezes a necessidade de redundância de dados no momento desta segmentação. Tanto a transformada de Hilbert quanto a FFT utilizaram um matriz como estrutura de dados, onde o número de linhas é igual ao número de antenas e o número de colunas igual ao tamanho do vetor de amostras das antenas, ou seja, o elemento de partição dos dados utilizado até este ponto foi a antena.

Contudo, o próximo estágio de processamento exige um ajuste nesta estrutura de dados. O processo de segmentação dos dados para realizar a multiplicação cruzada, não é mais a antena mas sim a linha de base formada por cada par de antenas. Neste ponto, a matriz resultante da FFT, Figura 3.19, matriz à esquerda, é transformada em um vetor coluna, onde cada posição possui uma matriz de duas linhas, formada pelo par de antenas de uma linha de base, e o número de colunas, segue o tamanho do vetor de amostra das antenas, como pode ser observado na Figura 3.19, à direita. O objetivo é que cada posição do vetor coluna, represente uma linha de base, isso faz com que seja necessário um bloco funcional para a formatação dos dados referentes à linha de base, seguindo a estratégia de processamento, dentro da plataforma CoP-WS.

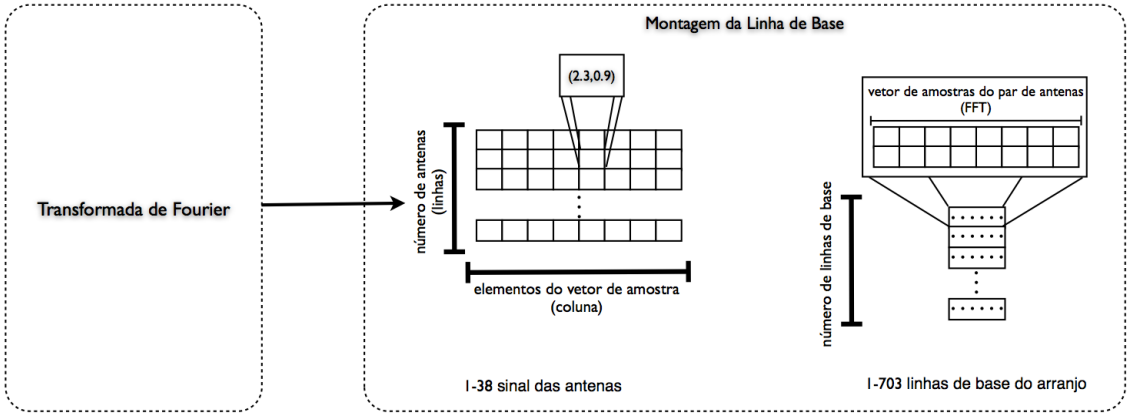


Figura 3.19: Representação da estrutura de dados de entrada e saída do bloco funcional de Montagem de Linhas de Base.

3.3.2 Reconhecimento de explosões solares

A segunda aplicação implementada para prova de viabilidade da plataforma CoP-WS teve como o objetivo realizar o processo de reconhecimento de explosões solares. Esta aplicação recebe um conjunto de imagens, que são resultantes da observação realizada por um determinado instrumento, e utilizando estas imagens, faz o processamento com o objetivo de reconhecer um padrão específico. Neste caso específico as imagens produzidas são do disco solar, e o padrão a ser reconhecido são de explosões que ocorreram durante a observação do Sol.

Como a plataforma CoP-WS não se encontrava conectada a um instrumento de forma direta para realizar a obtenção das imagens, foi utilizado um conjunto de imagens produzidas pela observação do Rádio-Heliógrafo de Nobeyama no Japão (NRO, 2012). Maiores informações sobre o Rádio-Heliógrafo de Nobeyama são descritas por Nakajima e seus colaborares (NAKAJIMA et al., 1994). As imagens do disco solar produzidas pela observação deste instrumento, foram colocadas em um mesmo diretório, onde, para o processamento, o primeiro nó do arranjo buscava as imagens dentro de uma ordem, com o objetivo de simular um fluxo de envio das imagens.

Como metodologia para o processamento de reconhecimento de explosões solares foram utilizados o GLCM e a teoria de Near Sets, apresentadas nas Seções 2.14 e 2.13 respectivamente. A escolha pelo uso desta abordagem se deve a dois fatores, primeiro pela questão do Near Sets não ter sido utilizado para o reconhecimento de explosões solares, e segundo pelas características do Near Sets, de definir funções de descrição baseada em valores reais, permite uma implementação mais adequada para seu processamento dentro da arquitetura do GPU/-CUDA.

As imagens do disco solar de Nobeyama estão no formato RGB e possuem uma dimensão

de 500x500 pixels, desta forma o primeiro passo foi a conversão destas para tons de cinza. Cada uma destas imagens em tons de cinza foram divididas em sub-imagens de 50x50 pixels de dimensão, sendo estas sub-imagens, chamadas de janelas, que foram utilizadas para a extração das características. A Figura 3.20 apresenta o processo de geração das janelas discutido, sendo apresentada do lado esquerdo superior, a imagem original do disco solar, abaixo desta imagem a mesma imagem convertida para tons de cinza, e a direita o mozaico da imagem do disco solar, em tons de cinza, formado pelas janelas geradas.

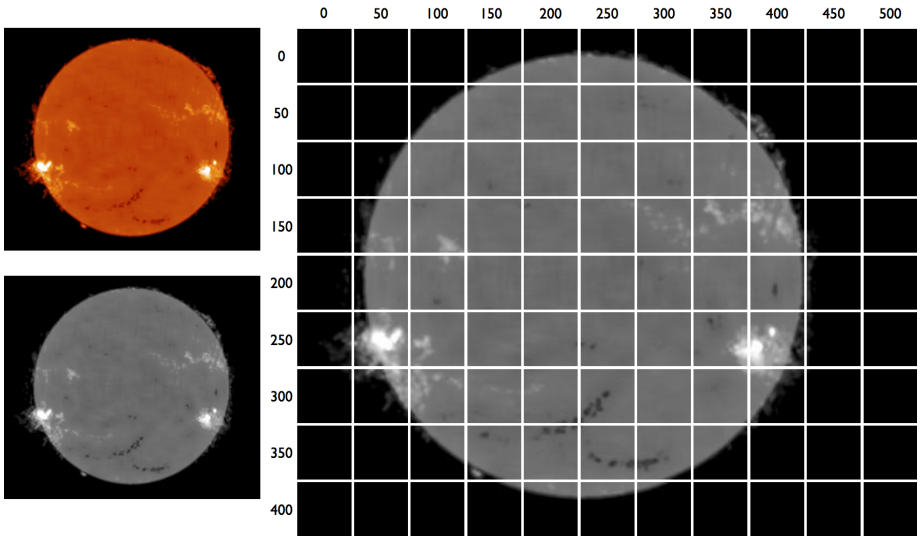


Figura 3.20: Processo de geração de janelas de 50x50 do disco solar (NOBEYAMA, 2013).

Diferente da aplicação do correlacionador discutida anteriormente, o processamento de reconhecimento de padrões de explosão solares, está organizado em duas fases que são: 1) o treinamento, onde a aplicação é treinada a reconhecer o padrão da explosão solar; e 2) o processamento de reconhecimento propriamente dito. Esta característica fez com que esta aplicação fosse implementada em dois aplicativos menores, o primeiro responsável pelo treinamento e o segundo para realizar o reconhecimento do padrão de explosão solar propriamente dito.

Salienta-se que apesar da necessidade dos dois processamentos, o aplicativo responsável pelo treinamento não foi implementado na plataforma CoP-WS, uma vez que apenas o resultado desse treinamento é usado no processamento do reconhecimento propriamente dito.

3.3.2.1 Treinamento

A aplicação de treinamento para o reconhecimento dos padrões de explosão solar (Figura 3.21), tem como entrada janelas, que são sub-imagens com dimensão 50x50 pixels, extraídas de uma imagem do disco solar, especialmente selecionadas, de forma a conter o evento típico de

uma explosão solar. Como estas janelas são extraídas diretamente da imagem original do disco solar produzida por Nobeyama, estas se encontram no formato RGB, desta forma o primeiro estágio de processamento está na conversão da janela de RGB para tons de cinza. Em seguida esta janela passa pelo processo de extração de características, segundo a teoria apresentada pelo GLCM, sendo considerados os valores de funções de descrição de Contraste, Correlação, Entropia , Energia e Homogeneidade dos pixels desta janela. Todo o conjunto de sub-imagens com padrões de explosão solar passam por estes processamentos, que no seu final gera uma matriz, onde o número de linhas é igual ao número de sub-imagens processadas e cada coluna refere-se a uma característica extraída usando o GLCM, ou seja, cinco colunas.

O próximo estágio do treinamento para reconhecimento dos padrões de explosão solar, utiliza a matriz resultante no estágio anterior, com as características GLCM obtido pelo processamento de todas as sub-imagens, para a criação de classes, tomando como base a similaridade apresentada entre cada uma das sub-imagens utilizadas. Este processo de verificação de similaridade utiliza como base a teoria do Near Sets e utiliza a tolerância de similaridade ε, na verificação entre os valores obtidos. Este valor de tolerância é obtido empiricamente em função dos melhores resultados obtidos. Cada conjunto de sub-imagens que apresentam uma distância de similaridade $\leq \varepsilon$, é agrupado em classe, onde os valores das funções de descrição desta classe é a média aritmética dos valores das funções de descrição de cada uma das sub-imagens da classe. O processo de classificação das sub-imagens (padrões), gera uma matriz, onde o número de linhas é igual ao número de classes definidas, e o número de colunas é o número de características utilizadas, e seus valores são utilizados como entrada na aplicação de reconhecimento dos padrões de explosão solar.

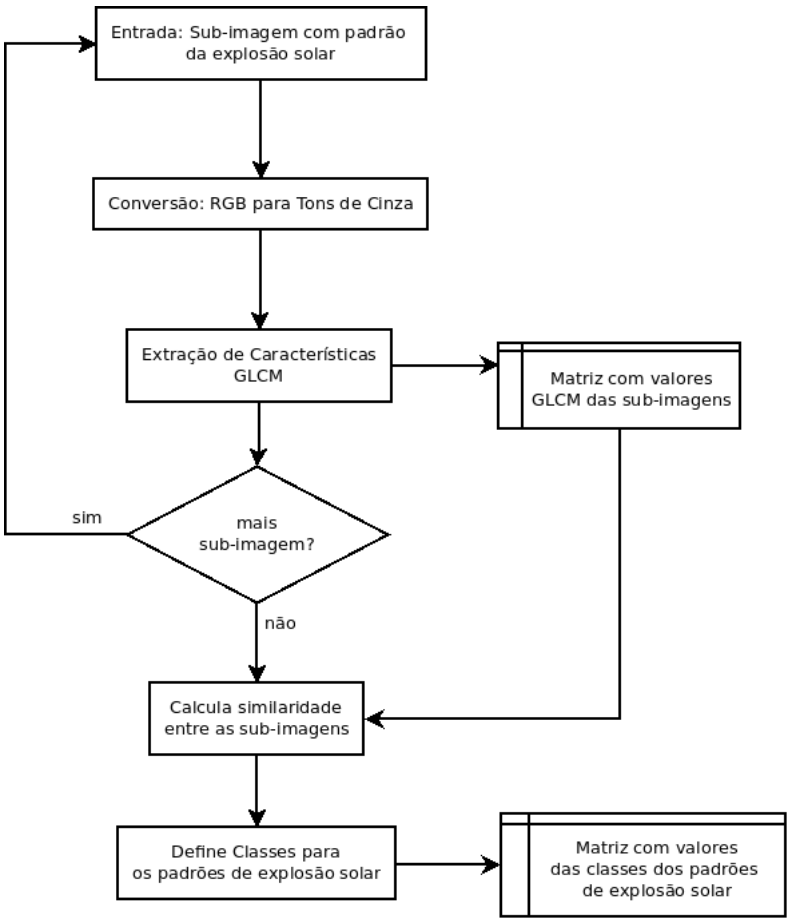


Figura 3.21: Fluxograma da aplicação de treinamento para o reconhecimento de padrões de explosão solar.

3.3.2.2 Reconhecimento

A aplicação para o reconhecimento de explosões solares (Figura 3.22) utiliza como entrada, um conjunto de imagens do disco solar. Em um ambiente real, estas imagens estariam sendo produzidas por um radiointerferômetro, durante a observação do Sol. Contudo na atual versão deste trabalho, como não foi possível a conexão com o instrumento BDA, e com o objetivo de simular um fluxo de dados, foram utilizadas imagens do disco solar, de observações realizadas pelo instrumento de Nobeyama (NRO, 2012). Uma vez que uma imagem do disco solar foi carregada, a primeiro estágio é a conversão desta imagem de RGB para Tons de Cinza, imagem esta que é utilizada para o processo de geração das sub-imagens.

Cada imagem do disco solar, em tons de cinza, é dividida em janelas de sub-imagens de 50x50 pixels, que são armazenadas em uma lista. Para cada uma destas sub-imagens é realizada a extração das características utilizando o GLCM, cujos valores são utilizados para calcular a distância da similaridade desta sub-imagens com cada uma das classes encontradas pelo processo de treinamento, processo no qual foi utilizado como valor de tolerância $\varepsilon = 0.1$. Caso a

sub-imagem seja classificada como sendo de uma explosão solar, a mesma é salva em um diretório específico, e em seu nome é utilizado o nome da imagem do disco solar, as coordenadas na janela, e a informação de qual classe esta foi classificada.

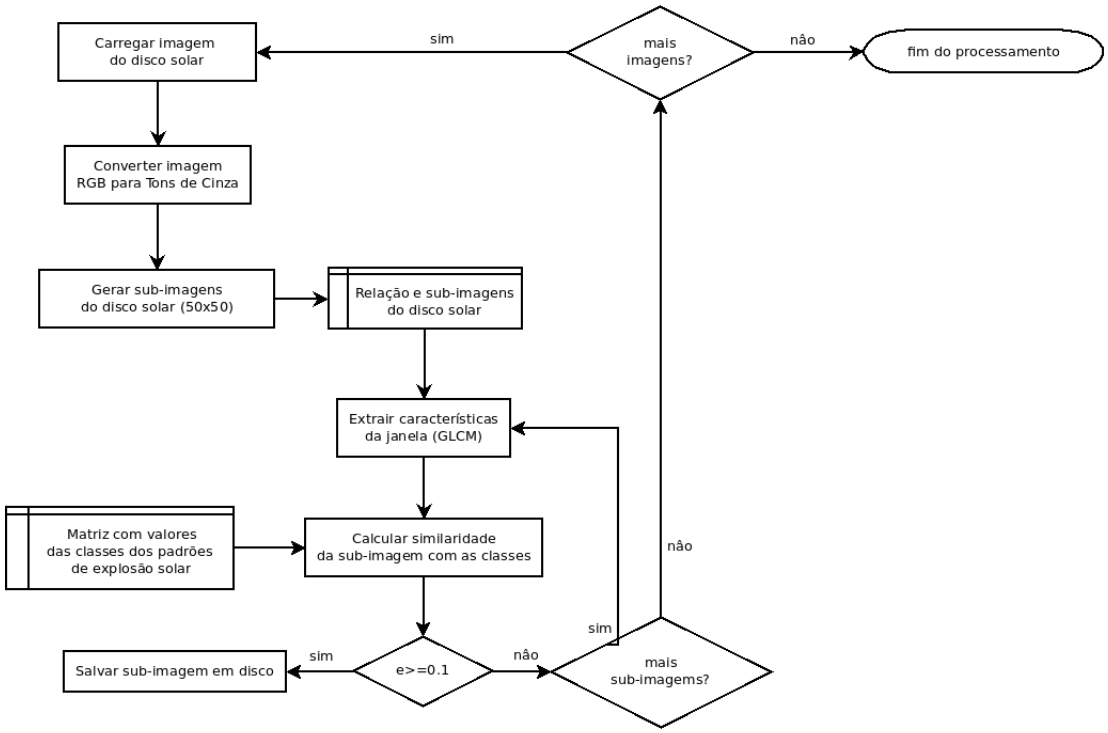


Figura 3.22: Fluxograma da aplicação o reconhecimento de explosões solares .

Capítulo 4

RESULTADOS

No capítulo anterior foi apresentada a plataforma CoP-WS, tendo sido discutidos os conceitos, a metodologia e as aplicações de validação. Neste capítulo são apresentados os resultados obtidos com a implementação da plataforma CoP-WS em si e dos resultados apresentados pelas duas aplicações de teste implementadas para testar a viabilidade da mesma.

O foco deste trabalho de doutorado é o desenvolvimento da plataforma CoP-WS, uma plataforma híbrida, de alto desempenho, para o processamento de grandes volumes de dados produzidos de forma contínua, apresentada no Capítulo 3. Para a apresentação dos resultados, este capítulo se encontra organizado em duas partes: discussão da implementação da plataforma apresentada na Seção 4.1, e os resultados do processamento das duas aplicações implementadas para testar a viabilidade, sendo os resultados do correlacionador e da aplicação de reconhecimento das explosões solares, apresentadas na Seção 4.2 e 4.3, respectivamente.

4.1 Medidas de Tempo da Plataforma CoP-WS

Como resultado do processo de implementação da plataforma CoP-WS, teve-se um ambiente formado por três nós computacionais, cada qual com um WS instalado. Para a implementação dos WS desta versão, foi utilizada a linguagem de programação C++, com o *toolkit* gSOAP (EGELEN, 2013), e o modulo mod_gsoap (ABERGER, 2010) para o servidor web Apache (APACHE, 2013a), a definição final do ambiente operacional pode ser vista na Tabela 4.1.

A limitação no número de computadores utilizados, apresentados na Tabela 4.1, principalmente na fase de testes da plataforma, fez com que fossem adotadas as seguintes técnicas para tomada de dados e análise do resultados:

Nó	Processador	Memória RAM	CoProcessador	S.O.
0	CPU i7 2.6 GHz	16 GB DDR3	GeForce GT 650M 1GB	Mac OS X 10.8.4
1	Quad Core 2.33 GHz	4 GB	GTX 295 1GB	Ubuntu 12.04
2	Quad Core 2.33 GHz	4 GB	GeForce GTX 8800 860 MB	Ubuntu 12.04

Tabela 4.1: Composição do arranjo final da plataforma utilizada para teste.

1. O tempo de processamento foi obtido considerando:

   (a) GPU GetForce GT 650M; e

   (b) CPU i7 2.6 GHz

2. Para a análise do comportamento do fluxo de comunicação foram adotadas as seguintes configurações, onde *N* refere-se à cardinalidade (número) de nós computacionais:

   (a) 1:1 (de uma máquina para outra máquina) - foram considerados o Quad Core;

   (b) 1:N (de uma máquina para N máquinas) - do CPU i7 para Os dois Quad Cores; e

   (c) N:1 (de N máquinas para 1 máquina)- dos Quad Cores para o CPU i7.

3. Em todos os casos o CPU i7 além de fazer o papel de nó computacional, também faz o papel de controle recebendo as notificações de tempo dos demais nós.

4.2 Resultados da Aplicação 1 : Correlacionador FX para Sinais de Radiointerferometria

Uma das aplicações implementadas para realizar o teste de viabilidade do uso da plataforma CoP-WS foi o processamento da correlação dos sinais de um arranjo interferométrico. Esta aplicação teve como inspiração o BDA (Seção 2.16), e o apoio do DAS (Divisão de Astrofísica) do INPE, responsável pelo projeto do BDA. Apesar do apoio interno do DAS e do fornecimento de dados do projeto/instrumento, não foi possível realizar testes com esta aplicação utilizando dados reais do BDA, seja fazendo a captura dos dados do arranjo das antenas de forma direta, ou mesmo utilizando dados arquivados em modo texto por exemplo. Essa dificuldade está relacionada com a falta de uma interface para a conversão dos dados das antenas para a plataforma CoP-WS.

Desta forma, foi implementado um sistema gerador de sinais que tem por objetivo a criação de uma massa de dados compativel com o volume gerado pelo instrumento BDA durante o período de observação de uma fonte. Este sistema não tem a pretenção de gerar sinais coerentes

e relacionados com uma fonte/evento de rádio específica, uma vez que os valores produzidos são obtidos de forma aleatória, o que faz com que a massa de dados produzida seja suficiente para gerar um volume de trabalho capaz de avaliar o CoP-WS nos processos que envolvem a correlação, mas não permite que seja realizada a deconvolução deste sinal gerado (saída do correlacionador), uma vez que os dados não possuem qualquer tipo de comprometimento com uma fonte quanto a produzir uma imagem. O correlacionador implementado tem como objetivo realizar testes em três conceitos da plataforma CoP-WS: a escalabilidade dos blocos funcionais, quanto a inclusão de novos nós; o comportamento dos nós, organizados em blocos funcionais, quanto a questão do processamento utilizando o esquema *pipeline*; e o tempo de resposta para um fluxo de dados contínuo.

Na Tabela 4.2 são apresentados os resultados da simulação do processamento dos dados durante os estágios de processamento da correlação dos sinais, considerando-se apenas o tempo de execução. Para se ter um quadro comparativo do desempenho das plataformas, foi implementado uma versão sequencial do correlacionador de sinais, e a plataforma CoP-WS foi configurada em dois arranjos, um utilizando um nó computacional para cada bloco funcional CoP-WS (*pipeline*), e outra com a paralelização nos blocos funcional responsáveis pela FFT e da multiplicação cruzada, com 6 e 5 nós computacionais respectivamente CoP-WS (*pipeline* paralelo). Em todos os casos foi considerado um arranjo onde a frequência de amostragem dos sinais das antenas é de 5 MHz, consequentemente a taxa de amostragem, considerando o teorema de Nyquist é de 10 MHz.

Na Tabela 4.2 a coluna "Memória"refere-se à quantidade alocada de memória em MB; "Tempo" refere-se ao tempo de execução do processamento, sem considerar o tempo de comunicação; e "Nós" refere-se ao número de nós computacionais alocados.

	Hilbert			FFT			Multiplicação Cruzada		
	Memória	Tempo	Nós	Memória	Tempo	Nós	Memória	Tempo	Nós
Sequencial	152	6,651	1	304	966,290	1	5.624	530,710	1
CoP-WSp	152	1,029	1	304	118,98	1	5.624	12,801	1
CoP-WSpp	152	1,029	1	304	29,831	6	5.624	3,551	5

Tabela 4.2: Medição do Tempo (ms) e Consumo de Memória (MB) da aplicação do correlacionador implementado.

Conforme descrito na Seção 4.1, devido a limitação do número de máquinas existentes, os resultados descritos na Tabela 4.2, foram obtidos por meio não de um processamento completo, mas realizando cada estágio do processamento de forma individual seguindo a ordem de exe-

cução da aplicação. Na terceira linha da Tabela 4.2 (CoP-WSpp) a coluna Nós apresenta uma quantidade sugerida de estações dentro do Bloco Funcional. Este número foi obtido por meio do estudo do comportamento dos nós computacionais em um mesmo bloco funcional.

Nas Tabelas 4.3,4.4 e 4.5 são apresentados quadros comparativos considerando a latência da rede, tempo de processamento e o speedup entre os três ambientes de processamento, sequencial, CoP-WSp com apenas um nó computacional por bloco funcional e uma projeção do CoP-WSpp com uma configuração formada por 12 nós computacionais, organizados por blocos funcionais,.

Nota-se que, em todos essas quatro tabelas, o speedup apresentado para CoP-WSp é em relação à arquitetura sequencial e o speedup apresentado para CoP-WSpp é em relação à arquitetura CoP-WSp.

	Hilbert		
	Comunicação (Rede)	Processamento (CPU/GPU)	*Speedup*
Sequencial	14,6	6,7	-
CoP-WSp	14,6	1,0	1.7
CoP-WSpp	14,6	1,0	1

Tabela 4.3: Tempo do processamento (ms), latência da rede (ms) e *speedup* do bloco funcional Hilbert.

	FFT		
	Comunicação (Rede)	Processamento (CPU/GPU)	*Speedup*
Sequencial	29,8	966.2	-
CoP-WSp	29,8	118,9	6.7
CoP-WSpp	4,9	29,8	4.3

Tabela 4.4: Tempo do processamento (ms), latência da rede (ms) e *speedup* do bloco funcional FFT.

	Multiplicação Cruzada		
	Comunicação (Rede)	Processamento (CPU/GPU)	*Speedup*
Sequencial	551.5	530.7	-
CoP-WSp	551.5	12,8	1.9
CoP-WSpp	110,3	3,55	4.9

Tabela 4.5: Tempo do processamento (ms), latência da rede (ms) e *speedup* do bloco funcional da Multiplicação Cruzada.

Nas Figuras 4.1, 4.2 e 4.3 são apresentados gráficos com a análise do consumo do GPU Geforce 650M durante a execução da aplicação. Foi utilizado um número total de 32 blocos de , cada qual com 256 threads (indicado pelo ponto vermelho), o que representa um número

aproximado de 8 mil threads sendo executada por cada ciclo de processamento, e uma eficiência aproximada de 0.71 e 0.98 para os bloco funcionais da FFT e da multiplicação cruzada respectivamente.

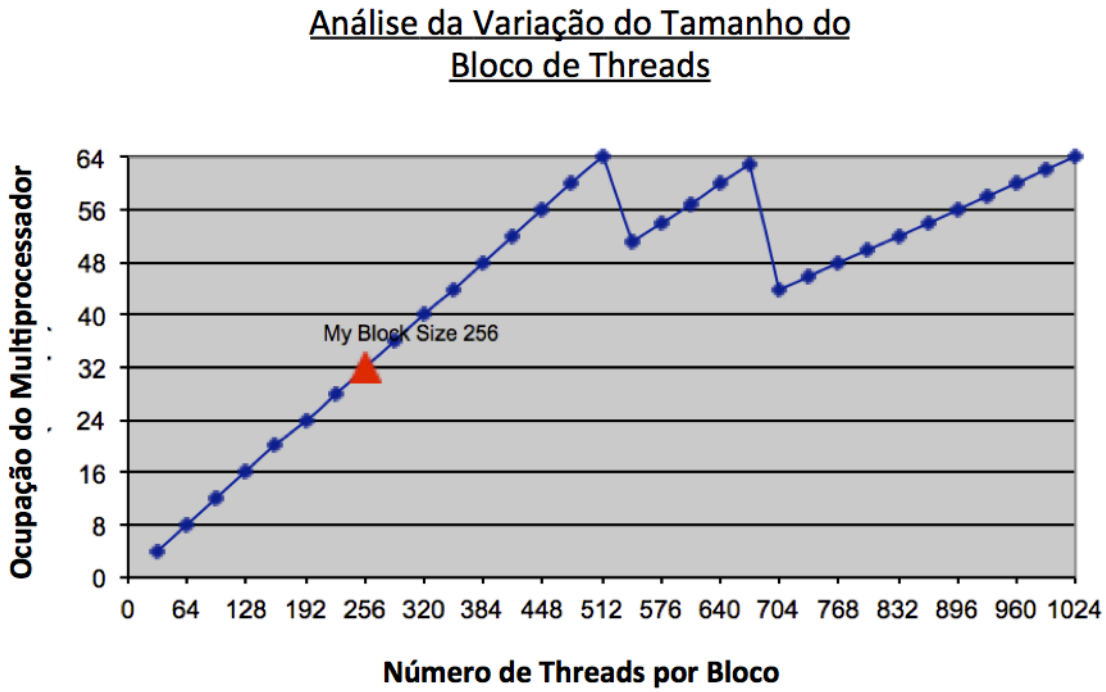


Figura 4.1: Análise da variação do tamanho do bloco de threads.

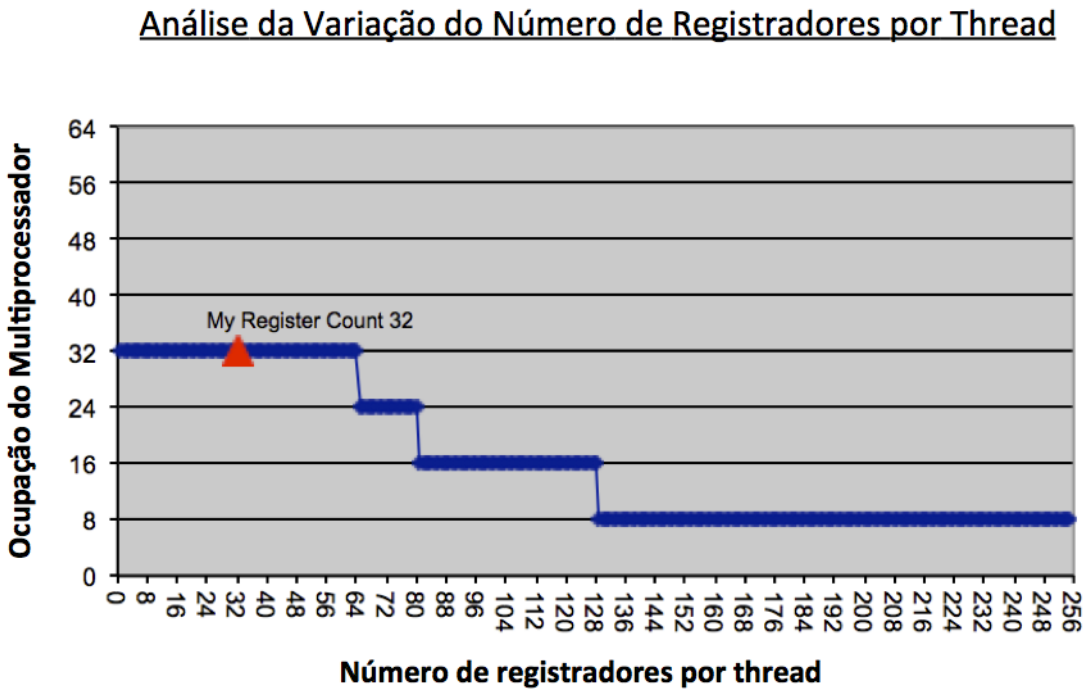


Figura 4.2: Análise da variação do número de registradiroes por threads.

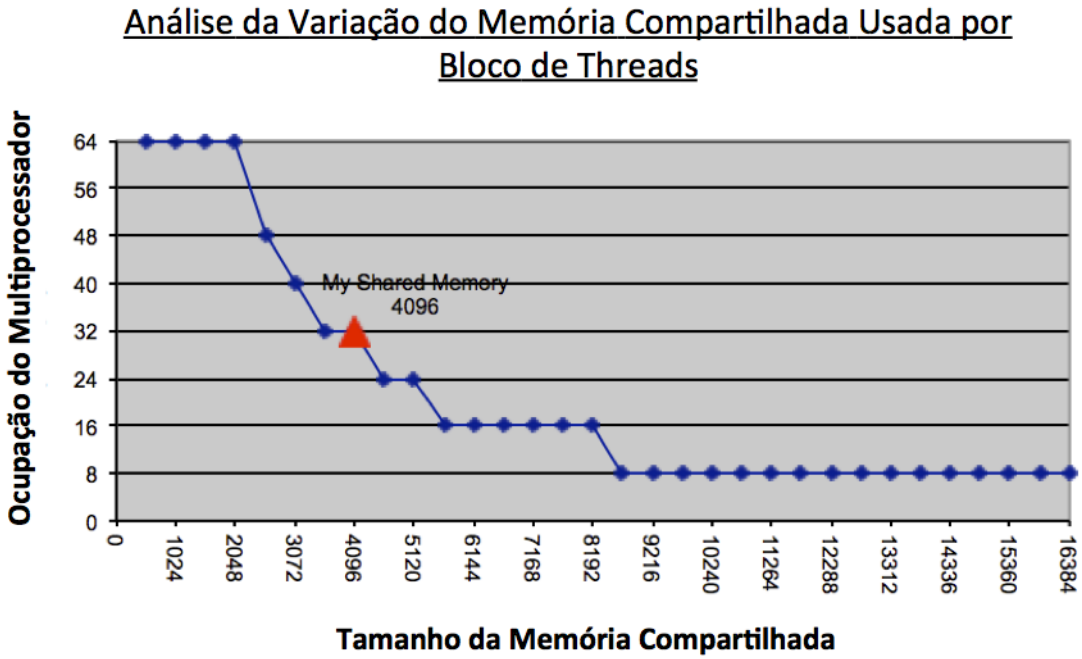


Figura 4.3: Análise da variação da memória compartilhada usada pelos blocos de threads.

Para determinar o número de nós computacionais que seriam utilizados em cada um dos blocos funcionais (estágios do processamento) foram levados em consideração duas variáveis: a quantidade de memória utilizada e o tempo máximo de processamento. Desta forma, como pode ser observado na Tabela 4.2, para um arranjo com 38 antenas, e para atender a um tempo de integração próximo a 100 ms, é necessário um total de 12 nós computacionais, estando estes dispostos no arranjo 1 nó para o processamento da Transformada de Hilbert, 6 nós para o processamento da FFT, e 5 nós para a multiplicação cruzada, com a necessidade de um ultimo nó para realizar a integração dos dados produzidos pela multiplicação cruzada. Uma representação gráfica da disposição destes nós dentro do arranjo computacional pode ser vista na Figura 4.4, com os quatro estágios do processamento.

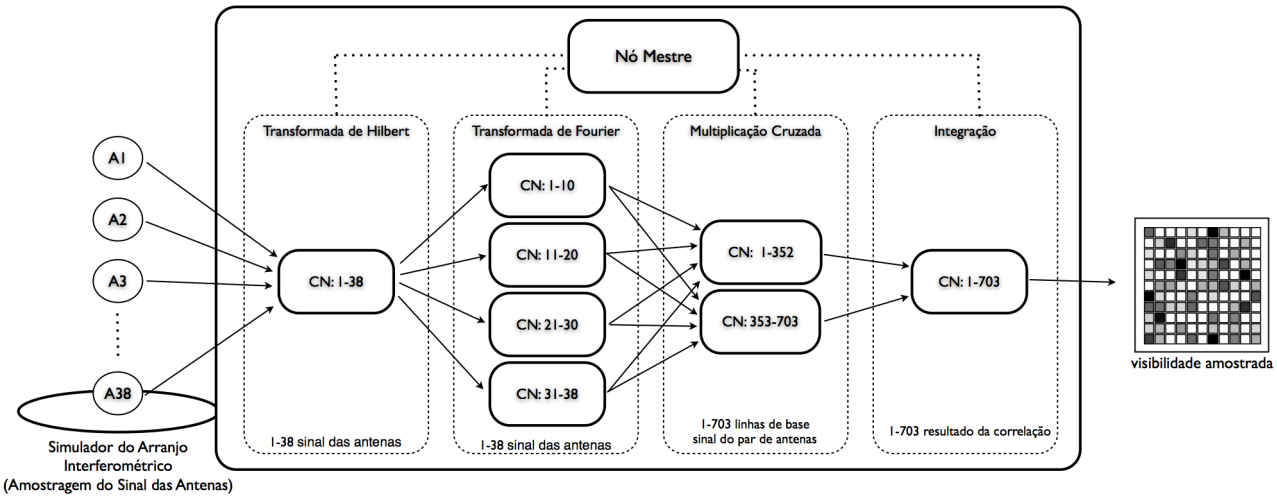


Figura 4.4: Processo de implantação do correlacionador na plataforma CoP-WS.

Com a configuração do arranjo computacional apresentada na Figura 4.4, o tempo total de processamento para a correlação do sinal ficou em ~164 ms, levando em consideração o tempo de latência da rede e utilizando uma média dos resultados obtidos durante a execução das simulações.

4.3 Resultados da Aplicação 2: Reconhecimento de Explosões Solares

A segunda aplicação implementada para prova de viabilidade da plataforma CoP-WS teve como objetivo realizar o processo de reconhecimento de explosões solares. Esta aplicação recebe um conjunto de imagens, que são resultantes da observação realizada por um determinado instrumento, e utilizando estas imagens, faz o processamento com o objetivo de reconhecer um padrão específico. Neste caso específico as imagens utilizadas são do disco solar, e o padrão a ser reconhecido são de explosões que ocorreram durante o imageamento do Sol.

Conforme mensionado anteriormente, como a plataforma não se encontrava conectada a um instrumento de forma direta para realizar a obtenção das imagens, foi utilizado um conjunto de imagens produzidas pela observação do instrumento de Nobeyama no Japão (NRO, 2012). As imagens do disco solar produzidas pela observação deste instrumento, foram colocadas em um mesmo diretório, onde, para o processamento, o primeiro nó do arranjo buscava as imagens dentro de uma ordem, com o objetivo de simular um fluxo de envio das imagens.

Como metodologia para o processo de reconhecimento de explosões solares foram utilizados o GLCM e a teoria de Near Sets, apresentadas nas Seções 2.14 e 2.13 respectivamente.

As imagens do disco solar de Nobeyama estão no formato RGB e possuem uma dimensão de 500x500 pixels, desta forma o primeiro passo foi a conversão destas para tons de cinza. Cada uma destas imagens em tons de cinza foram divididas em sub-imagens de 50x50 pixels de dimensão, sendo estas sub-imagens, chamadas de janelas, que foram utilizadas para o processamento.

4.3.1 Processo de treinamento

O primeiro passo para o processamento do reconhecimento de explosões no disco solar, é a realização do treinamento do sistema. Este processo de treinamento consiste em "ensinar" o sistema o que é a definição de uma explosão solar, de forma que este possa procurar por padrões similares durante a fase de execução da aplicação. Na Figura 4.5 podem ser observados os padrões de treinamento utilizados para o processo de treinamento da aplicação de reconhecimento de explosões solares, geradas por observações realizadas pelo instrumento de Nobeyama nos dias 1 (Pad1), 4 (Pad2 e Pad3), 5 (Pad 4 e Pad5) e 6 (Pad6 e Pad7) de Julho de 2012, em que na parte superior da pode ser visto a imagem original do disco solar, e abaixo da imagem de cada disco as janelas com as explosões solares que serão utilizadas para o processo de treinamento.

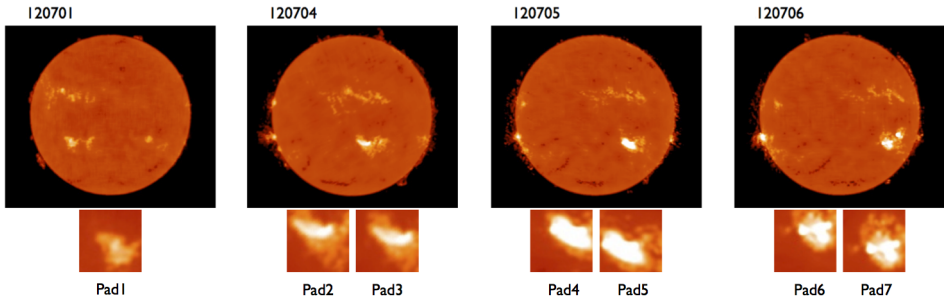


Figura 4.5: Padrões de treinamento.

Cada uma destas janelas passou pelo processo de conversão de RGB para tons de cinza para então ser utilizada como entrada no módulo de processamento do GLCM, que é o responsável pela processo de definir as características encontradas em cada um das explosões selecionadas. Foram utilizadas como funções de descrição os valores de Contraste, Correlação, Energia, Entropia e Homogeneidade, vistos na Tabela 4.6, onde para cada característica são apresentados os valores sem normalizar (A) e os valores normalizados (N).

	Contraste (ϕ_1)		Correlação (ϕ_2)		Energia (ϕ_3)		Entropia (ϕ_4)		Homogeneidade (ϕ_5)	
	A	N	A	N	A	N	A	N	A	N
Pad1	82190551.0	0.5318	4540590925.5	0.4451	493.0477	0.7932	-488879.8753	0.5717	17851.2750	0.7946
Pad2	102629225.0	0.6640	5803349290.0	0.6092	551.6865	0.8875	-643877.9087	0.7529	22398.9063	0.9971
Pad3	93027507.0	0.6019	5914622626.7	0.6209	537.8550	0.8653	-607166.7689	0.7100	21436.2540	0.9542
Pad4	113923128.0	0.7371	7159917409.4	0.7516	566.0044	0.9106	-693053.8819	0.8104	18973.3716	0.8446
Pad5	115425472.0	0.7468	8234637594.4	0.8644	587.1669	0.9446	-752254.2379	0.8796	21051.7919	0.9371
Pad6	125261634.0	0.8105	6024601298.5	0.6324	548.4597	0.8823	-640421.1374	0.7489	14743.7996	0.6563
Pad7	102344297.0	0.6622	7906861786.0	0.8300	563.6133	0.9067	-681220.8089	0.7966	22465.0252	1.0000

Tabela 4.6: Tabela com os valores GLCM obtidos.

O próximo passo realizado durante o treinamento foi a criação de classes de explosões solares. Onde cada uma destas classes é representada por um, ou mais sub-imagens, que apresente, segundo a teoria de Near Sets, uma proximidade quanto as suas características, ou seja, que apresentem similaridade entre os padrões, tendo sido utilizado $\varepsilon = 0.1$ para essa verificação. Na Figura 4.6 é apresentada a classificação de cada um deste padrões em suas Classes: A (Pad1), B (Pad2 e Pad3), C (Pad4), D (Pad5), E (Pad6) e F (Pad7). O valor da funções de descrição de cada uma das classes é a média aritmética das funções de descrição do conjunto de janelas participantes de cada um destas.

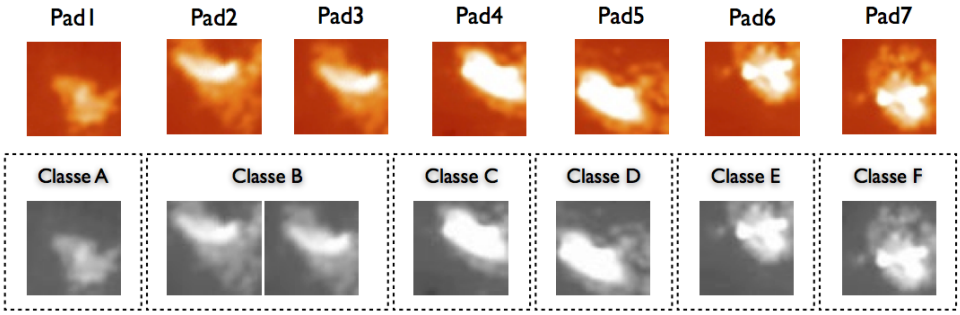


Figura 4.6: Padrões de treinamento.

A Tabela 4.7 apresenta os valores da norma L^2 da distância entre os padrões utilizadas para o treinamento, onde pode ser observado a proximidade entre os padrões Pad2 e Pad3 com um valor de 0.0904 (em destaque).

4.3.2 Processo de reconhecimento

Uma vez que as classes de explosões solares tenham sido definidos, a próxima etapa é o processamento de reconhecimento das explosões solares propriamente dito. Para isso foram utilizados novos padrões de disco solar resultado das observações realizadas pelo instrumento de Nobeyama nos dias 07, 08, 09, 10, 11, 12, 13 e 14 de Julho de 2012 (Figura 4.7). Estas imagens foram colocadas num diretório identificado dentro do nó computacional de entrada da

	Pad1	Pad2	Pad3	Pad4	Pad5	Pad6	Pad7
Pad1	0.000						
Pad2	0.3566	0.0000					
Pad3	0.2926	**0.0904**	0.0000				
Pad4	0.4575	0.2296	0.2439	0.0000			
Pad5	0.6142	0.3248	0.3546	0.1795	0.0000		
Pad6	0.4138	0.3717	0.3663	0.2442	0.4091	0.0000	
Pad7	0.5204	0.2259	0.2422	0.1900	0.1497	0.4267	0.0000

Tabela 4.7: L^2normalizado entre os pares de sub-imagens do treinamento.

plataforma, e sua rotina de processamento buscava uma imagem em sequência desta relação. Este procedimento simula a recepção das imagens produzidas por um instrumento durante o seu processamento da observação de um evento.

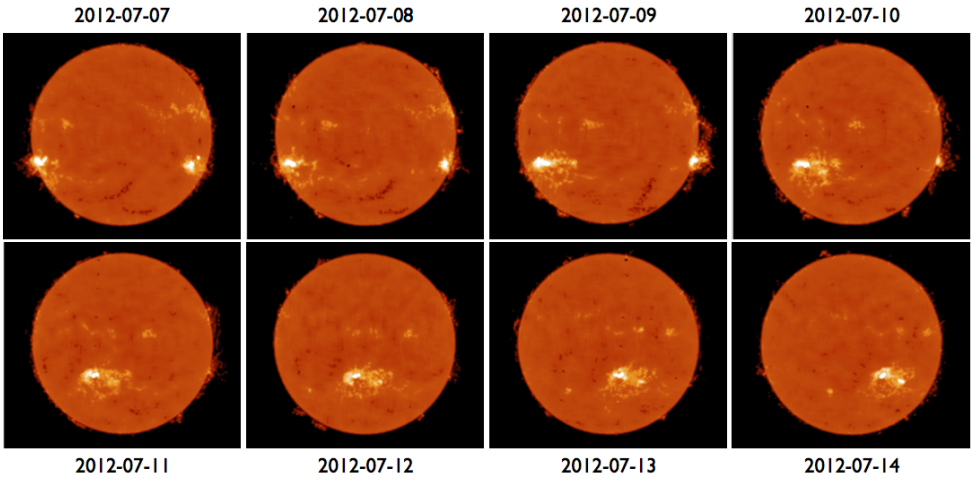


Figura 4.7: Imagens do disco solar usados para reconhecimento.

O fluxo de processamento para o reconhecimento de explosões solares pode ser visto na Figura 4.8, onde cada uma das imagens encontradas no diretório foi utilizada como entrada do sistema que apresentou como estágios de processamento: a conversão da imagem de RGB para tons de cinza; a geração de sub-imagens; o processo de extração das funções de descrição do GLCM de cada uma das sub-imagens; e o uso da Teoria de Near Sets para definir a similaridade de cada uma das sub-imagens com uma classe de explosão solar produzida no estágio de treinamento. As sub-imagens que apresentavam similaridade com as classes de explosões solares eram salvas em um diretório específico, tendo como parte de seu nome, o disco solar que pertencia, e qual a coordenada da sub-imagens dentro da imagem original.

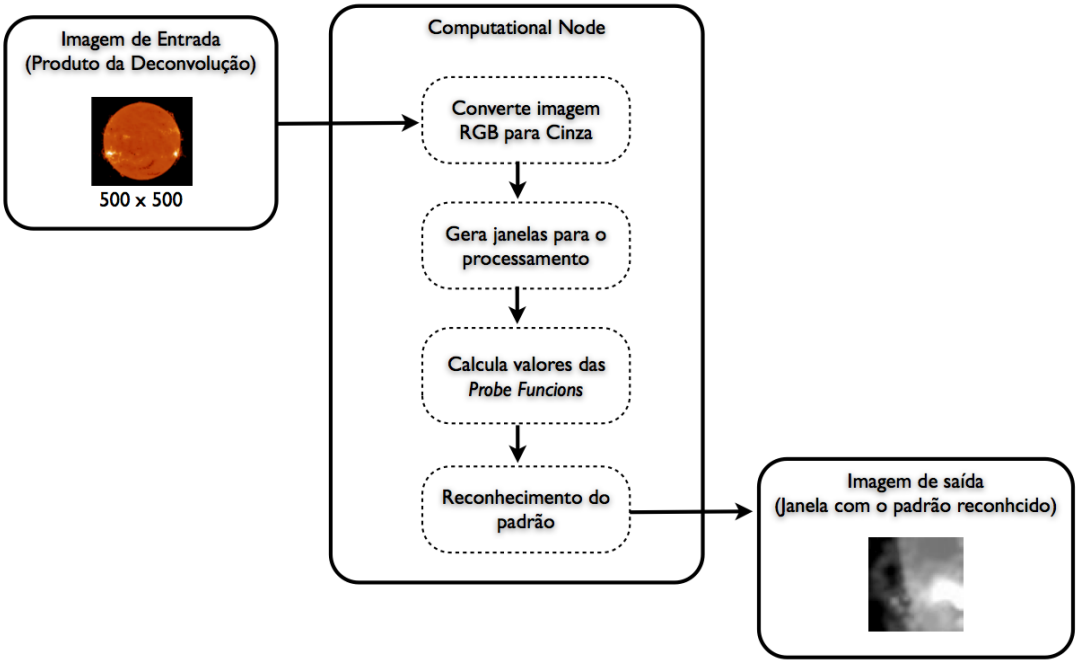


Figura 4.8: Fluxo de processamento para reconhecimento da explosão solar.

Na Figura 4.9 é apresentado o resultado da execução da aplicação de reconhecimento de explosões solares, onde, do lado direito, podem ser vistas as 8 imagens de disco solar utilizadas como entrada. A área do disco solar onde é encontrada a explosão (flare), encontra-se destacada por meio de um círculo enumerado, de forma que, dentro das 8 imagens, existe um total de 12 flares que podem ser reconhecidos.

Do lado direito da Figura 4.9 são apresentadas as sub-imagens reconhecidas como flares, organizadas em uma tabela, em que as linhas são as imagens de entradas, indicadas pela data da observação, e as colunas as classes de padrão de flare geradas na fase de treinamento. Caso o processo de reconhecimento da sub-imagem como sendo um flare, seja verdadeiro e correto, ao lado da sub-imagem pode ser visto o número que referência ao flare, caso contrário (sub-imagem reconhecida como flare, mas de forma incorreta), a sub-imagem é apresentada, mas sem o número de referência do flare.

Das 12 ocorrências de flares, distribuídas nas 8 imagens de disco solar, apenas o flare 8, da observação do dia 2012-07-10, não foi reconhecido, o que representa uma taxa de acerto de 91,6%, com um taxa de reconhecimento de falsos flares de 23%, já que das 30 sub-imagens reconhecidas como flare, 7 são falsos flares. Na Tabela 4.8 é apresentada a relação de *flares* reconhecidos e as respectivas classes.

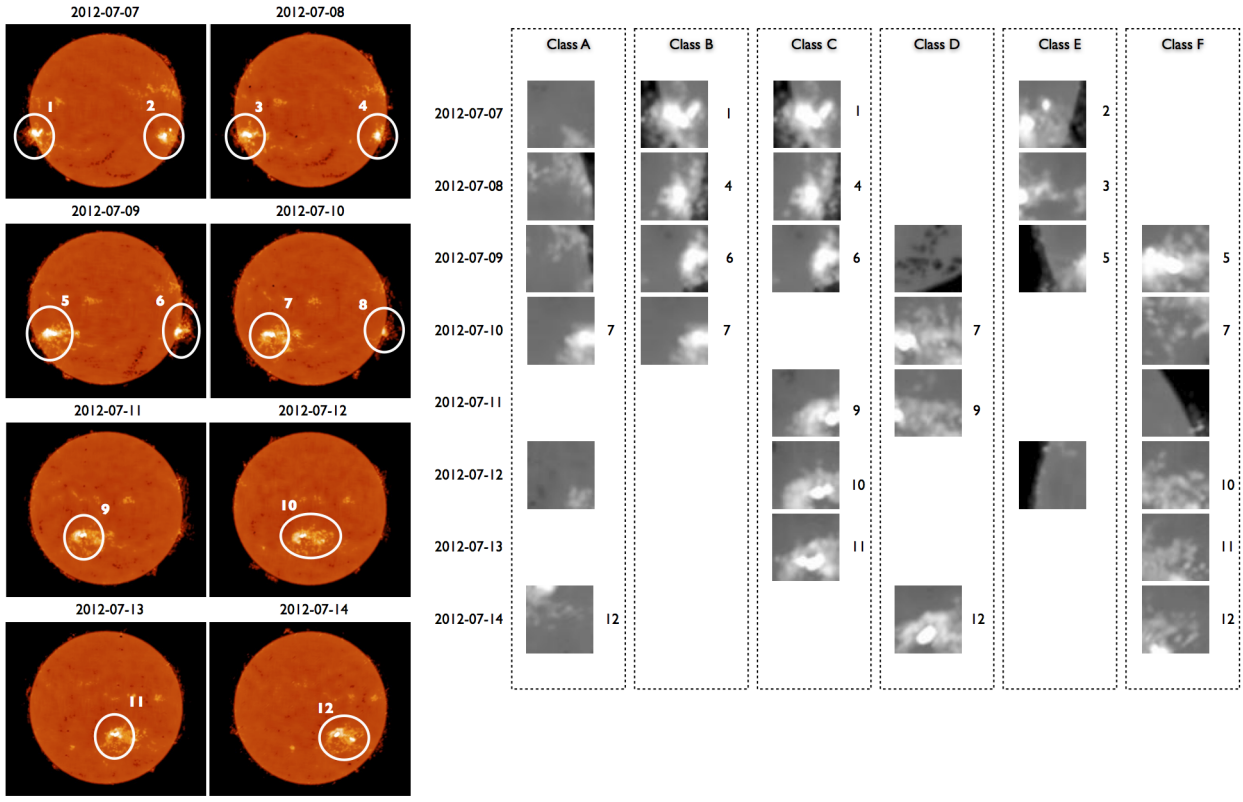


Figura 4.9: Padrões de explosão solar reconhecidos.

Classe	*FLARES*											
	1	2	3	4	5	6	7	8	9	10	11	12
A							X					X
B	X			X		X	X					
C	X			X		X			X	X	X	
D							X		X			X
E		X	X		X							
F					X		X			X	X	X

Tabela 4.8: Padrões reconhecidos pela aplicação.

Na Tabela 4.9, pode ser visto o tempo de processamento total que foi de 20 ms utilizando a plataforma CoP-WS de um nó computacional contra 2033.9 ms para um processamento sequencial em um CPU i7. Usando dois nós computacionais o resultado foi de 10.2 ms conforme esperado, e melhores do que os apresentados na Tabela 1.1.

Processador(s)	Memória Alocada (KB)	Tempo Processamento (ms)	Nós
Sequencial		2033.9	1
CoP-WSp	720	20.4	1
CoP-WSpp	360	10.2	2

Tabela 4.9: Tempo de processamento para reconhecimento das explosões solares.

Nas Figuras 4.10, 4.11 e 4.12 são apresentados gráficos com a análise do consumo do GPU Geforce 650M durante a execução da aplicação. Foi utilizado um número total de 16 blocos de threads, cada qual com 128 threads (indicado pelo ponto vermelho), o que representa um número aproximado de 2 mil threads sendo executada por cada ciclo de processamento.

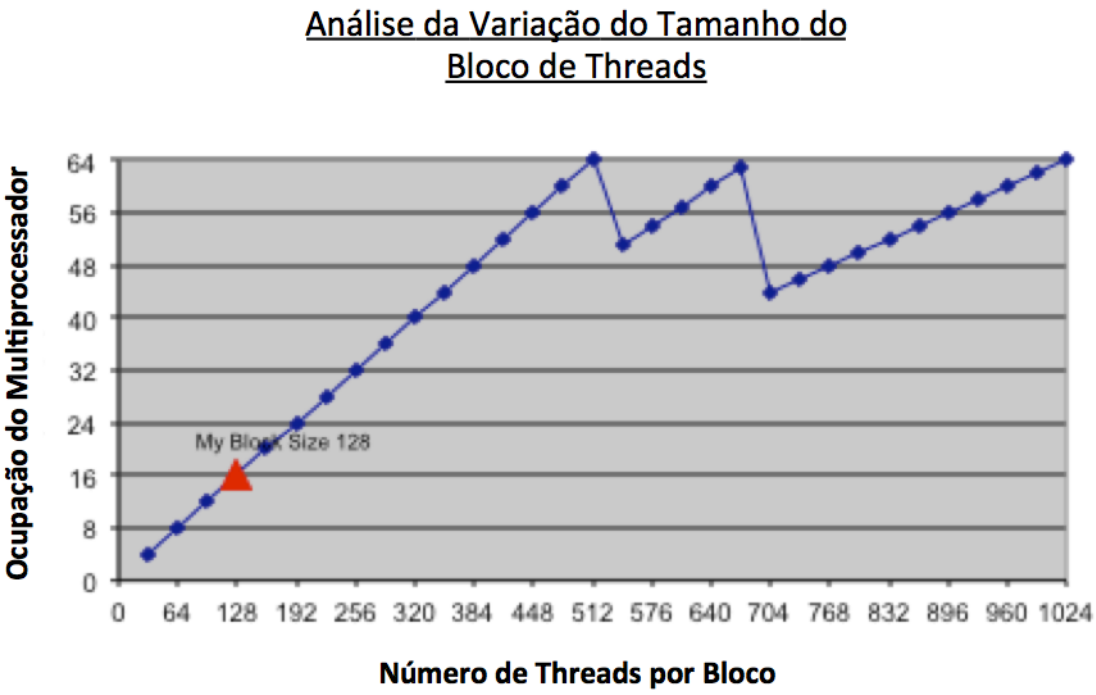


Figura 4.10: Análise da variação do tamanho do bloco de threads.

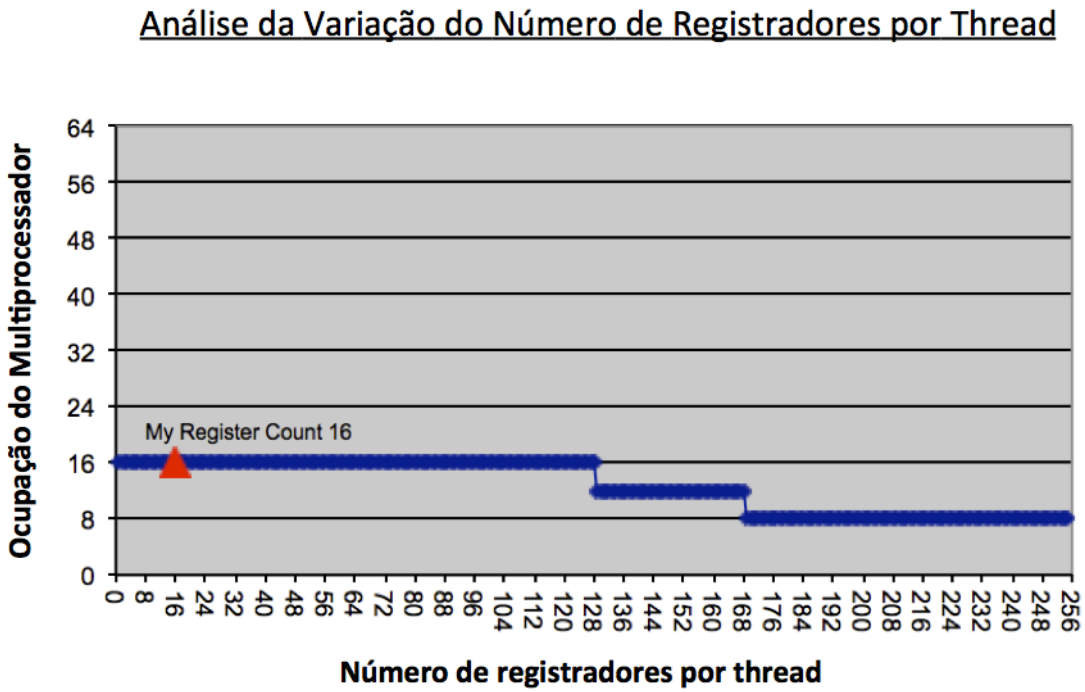


Figura 4.11: Análise da variação do número de registradiroes por threads.

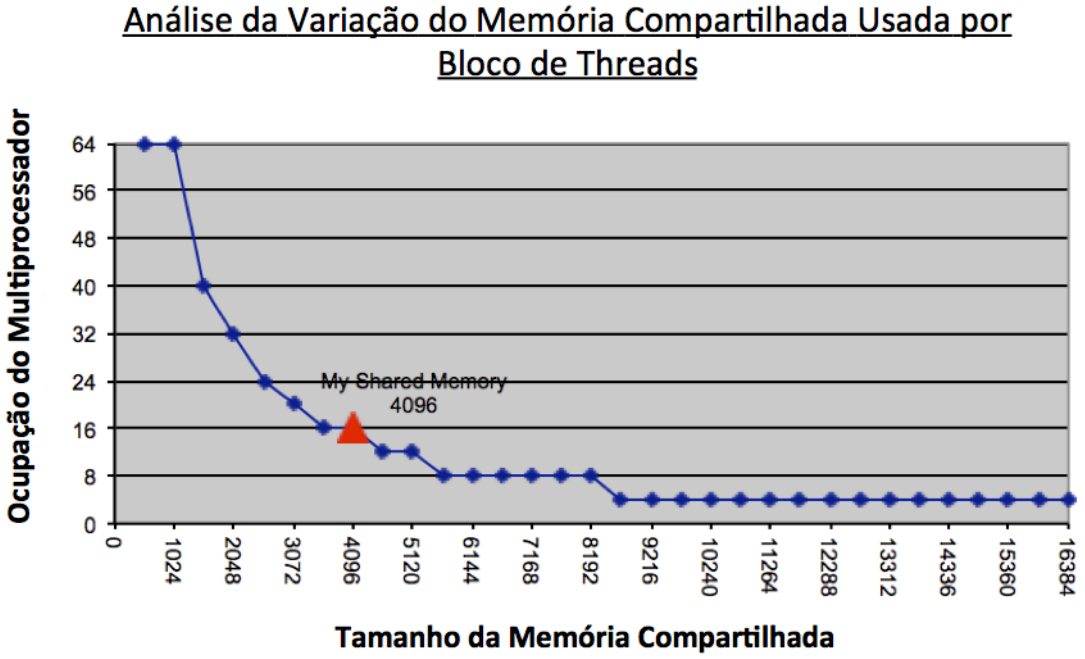


Figura 4.12: Análise da variação da memória compartilhada usada pelos blocos de threads.

As Figuras 4.13 e 4.14 apresentam gráficos comparativos entre pares das cinco funções de descrição utilizadas neste trabalho.

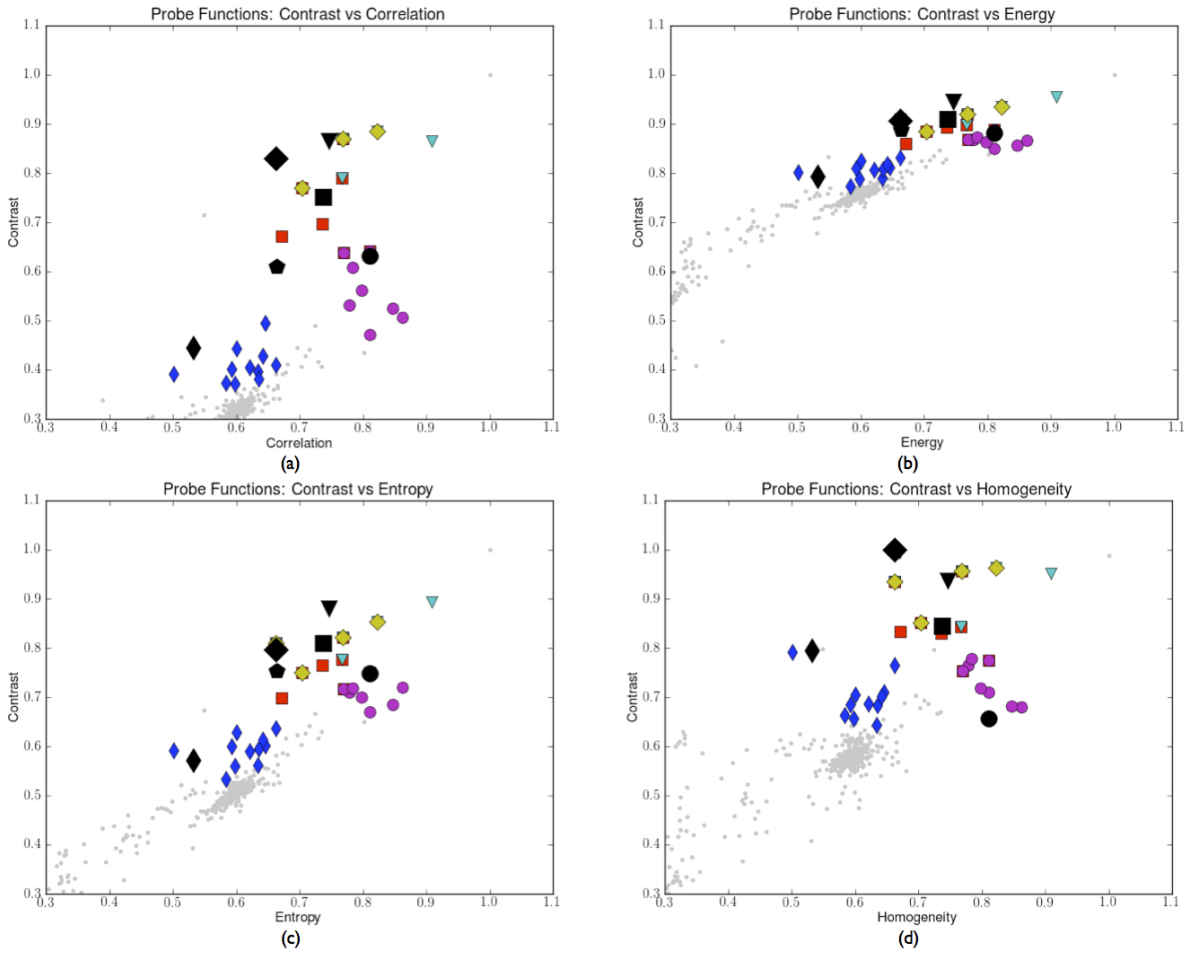


Figura 4.13: Gráficos comparativo das funções de descrição: Contraste vs Correlação (a), Energia (b), Entropia (c) e Homogeneidade (d).

Na Figura 4.13 são mostrados quatro gráficos comparativos entre a função de descrição contraste versus os outros quatro, ou seja, Figura 4.13(a), correlação, Figura 4.13(b), a energia, Figura 4.13(c), a entropia e Figura 4.13(d), a homogeneidade. Nos gráficos, cada uma das classes 7 de *flare* (A a F) é identificada por um símbolo diferente, como segue: A - diamante afiada, B - pentágono, C - quadrado, D - triângulo, E - círculo e F - diamante.

O representante de cada classe é desenhado com o mesmo símbolo (porém maior) daqueles que ele representa, pintado na cor preta. Os pontos cinza, em cada traço, representam sub-imagens que não foram reconhecidos. Pode-se notar que, em todos os gráficos, a sub-imagens não-reconhecidas, os valores da função de descrição menores do que os das sub-imagens reconhecidas. Isto é devido ao fato das funções de descrição que representam sub-imagens ter valores elevados. Pode-se notar que, em 4.13(a), 4.13(c) e 4.13(d), ou seja, as parcelas referentes ao contraste contra correlação, entropia e homogeneidade, respectivamente, existe uma maior grau de dispersão de dados em 4.13(b) , este tipo de dispersão promove uma melhor

separação entre as classes e, consequentemente, ajuda a identificar um grupo de funções de descrição mais adequado para discriminá-las.

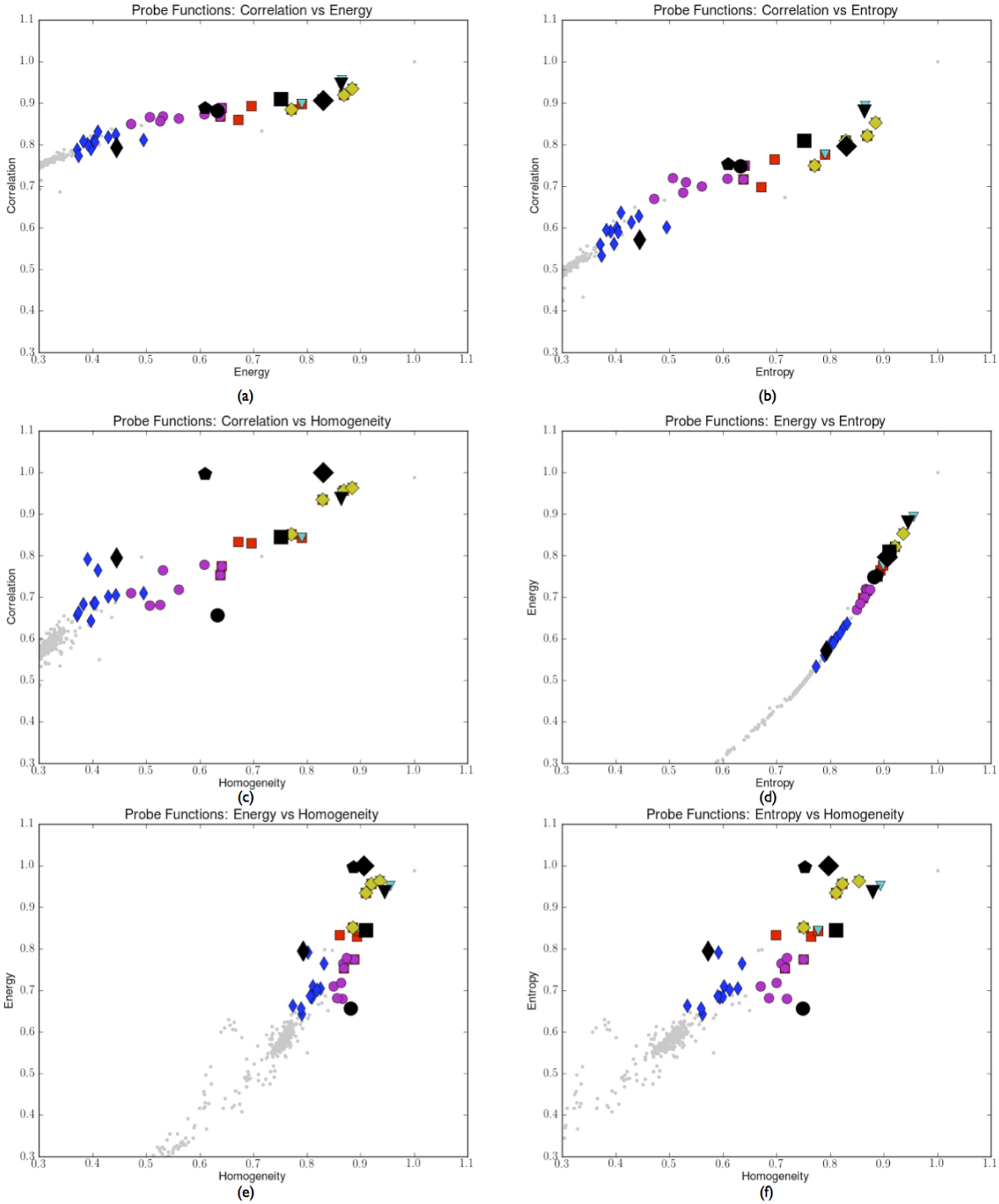


Figura 4.14: Seis gráficos comparativos entre as funções de descrição: Correlação vs Energia (a), Entropia (b), Homogeneidade(c), Energia vs Entropia (d), Homogeneidade (e) e Entropia vs Homogeneidade (f).

A Figura 4.14 também mostra gráficos dos restantes das quatro funções de descrição, iden-

tificando como eles se relacionam uns com os outros no experimento: Figura 4.14(a) correlação × energia, Figura 4.14(b) correlação × entropia, Figura 4.14(c) correlação × homogeneidade, Figura 4.14(d) A energia × entropia, Figura 4.14(e) a energia × homogeneidade e Figura 4.14(f), a entropia × homogeneidade. Os gráficos apresentados nas Figura 4.14(a) e Figura 4.14(b), mostrando a relação entre correlação × energia e correlação × entropia, respectivamente, em conformidade com o mesmo padrão, ou seja, os representantes de ambos, bem como as subimagens reconhecidas, siga o mesmo, quase linear, sequencial. Isto pode ser interpretado como as três funções de descrição a serem correlacionadas. A similaridade dos gráficos da Figura 4.14(c), Figura 4.14(e) e Figura 4.14(f) refletir o fato evidenciado em 9 (a) e 9 (b), ou seja, que a correlação, entropia e energia são correlacionadas.

Capítulo 5

CONCLUSÃO

No capítulo anterior foram apresentados os resultados obtidos com a utilização da plataforma CoP-WS, tendo sido implementados dois aplicativos para testar a viabilidade de uso da mesma, sendo a primeira aplicação um correlacionador de sinais para um arranjo interferométrico, e a outra um sistema de reconhecimento de explosões solares. Neste capítulo é apresentada uma discussão sobre os resultados obtidos, levando em conta não apenas os resultados, mas também a experiência no processo de utilização da plataforma. No final do capítulo é apresentado um conjunto de sugestões para a realização de trabalhos futuros em continuidade.

Este trabalho de doutorado está inserido no escopo de computação de alto desempenho, tendo como principal motivação o processamento de grandes volumes de dados produzidos de forma contínua (*data stream*). A solução proposta por este trabalho foi a criação de uma plataforma híbrida, quanto as arquiteturas utilizadas, e organizadas em nós computacionais especializados, onde cada nó é composto por um par CPU-Coprocessador. O coprocessador pode ser qualquer dispositivo que dê suporte ao CPU no processamento dos dados, ou mesmo na realização de outras atividades, como por exemplo a aquisição de dados. Na versão atual deste trabalho foi utilizado como coprocessador um GPU (*Graphic Processing Unit*) da NVIDIA, por meio do ambiente CUDA (*Computed Unified Device Architecture*), um *framework* desenvolvido pela NVIDIA para que aplicações genéricas (não gráficas), possam ser processadas utilizando a arquitetura GPU. Estes nós computacionais são organizados em arranjos, onde os nós são utilizados para configurar blocos funcionais especializados no processamento de um estágio dentro do fluxo de dados da aplicação implementada, e a interface de comunicação entre cada um desses nós é realizada por meio de *Web Services*.

Dentro deste escopo, e como pode ser observado nos Capítulos 3 e 4, este trabalho também pode ser considerado como uma evolução seguindo os trabalhos de Harris (HARRIS, 2009) e de

Woods (WOODS, 2010). Ambos apresentaram o GPU como sendo uma alternativa viável para a implementação de um correlacionador. Contudo, os autores focam na implementação "direta" do correlacionador dentro do GPU em uma única estação de processamento. Neste trabalho é apresentada uma alternativa para a utilização do GPU, organizando esta em pares com o CPU, segundo um arranjo que permita o processamento de um fluxo de dados em um pipeline definido. Um outro ponto de relevância apresentado neste trabalho é a questão da utilização de padrões de software na implementação do correlacionador em parceria com o GPU. A criação de classes que representam as antenas e linhas de base do arranjo, permite que o consumo do dispositivo de coprocessamento seja utilizado de forma mais otimizada quando obsevado em um ambiente distribuído. Isso se dá por que cada objeto instanciado de uma classe, carrega consigo todos dados necessários para o seu processamento, ou seja, não depende de quaisquer outros dados existindo uma auto suficiência para o seu processamento.

O CoP-WS apresenta uma proposta não usual para a utilização do WS, fazendo com que estes se apresentem como o elo de comunicação entre os estágios de processamento de um aplicativo completo. Dentro desta visão o WS não oferece um serviço pontual para uma estação cliente, mas sim para outros WS, e justamente a união dos vários serviços oferecidos pelo arranjo completa a execução da aplicação. Desta forma o WS não está sendo utilizado apenas na comunicação entre uma estação cliente e um servidor, mas sim como um elemento de manutenção na rota de comunicação para o processamentos dentro da uma cadeia de processamento entre os nós do arranjo.

Uma das características fundamentais da plataforma apresentada neste trabalho é a sua capacidade de processamento dentro de um *pipeline* reconfigurável, ou seja, é possível especializar um nó computacional para o processamento de uma determinada tarefa, e agrupar nós com a mesma especialidade em blocos funcionais. Com a capacidade de adicionar ou remover nós de um determinado bloco funcional, busca-se a atingir um melhor ponto de desempenho para aquele estágio do processamento, e como os nós de um bloco funcional trabalham com um conjunto de dados obtido em tempos diferentes, pode-se obter um melhor desempenho por meio do processamento em paralelo.

Uma conclusão que se pode chegar quando se trabalha com arquiteturas híbridas é que existem dois pontos principais de dificuldades: o perfil do desenvolvimento da equipe e a integração do resultado do processamento resultante por diferentes arquiteturas. O primeiro ponto, o perfil da equipe, se deve ao fato da qualificação de desenvolvedor que será responsável pela implementação da aplicação dentro deste tipo de ambiente. Quando se usa mais de uma arquitetura, por exemplo GPU e FPGA, é muito difícil ter um mesmo profissional qualificado

para implementar de forma adequada um aplicação nas duas arquiteturas. Cada uma das arquiteturas apresenta diferentes mecanismos de refinamento para seus algoritmos e/ou mesmo filosofias para modelagem e implementação de suas rotinas. O segundo ponto, a integração dos resultados, está relacionada com a questão de que cada uma das arquiteturas, normalmente, apresenta diferentes formatos de arquivos, ou mecanismos de acesso aos dados, o que representa um problema quanto a questão de integração dos dados processados por cada uma destas arquiteturas. Nem sempre existe compatibilidade entre duas arquiteturas/plataformas fazendo com que sejam implementados mecanismos para efetivar essa transferência de dados. A plataforma CoP-WS pode ser vista como sendo uma possível solução para ambos os problemas. Quanto ao primeiro ponto, a formação da equipe de desenvolvimento da aplicação, a filosofia de trabalho do CoP-WS permite que diferentes desenvolvedores trabalhem de forma individual, e em paralelo, uma vez que cada uma das funcionalidades é completamente independente da outra. Desta forma desenvolvedores com diferentes especialidades podem trabalhar de forma independente, seguindo as especificações da API definida pelo projeto. Desta forma cada nó, *a priori*, poderia utilizar qualquer tipo de dispositivo como coprocessador, e este poderia ser implementado por um especialista dentro de sua área de conhecimento. Já quanto ao segundo ponto, relacionado com a questão da troca de dados entre diferentes plataformas, a plataforma CoP-WS se apresenta como solução pelo fato de que cada um dos nós é implementada de forma especializada. Desta forma um nó não sabe como é a implementação, ou mesmo a configuração, do nó que lhe antecede ou lhe sucede, havendo apenas a troca de dados dentro do fluxo de processamento.

Os resultados obtidos e apresentados no Capítulo 4, mostram que ambas as aplicações implementadas, o correlacionador de sinais de arquitetura FX, para arranjo interferométrico, e o sistema de reconhecimento de explosões solares, apresentaram um bom desempenho, enquanto que o processamento, atendia às especificações/requisitos. Um outro ponto interessante a ser observado, de forma mais clara na implementação do sistema de correlacionador, é o tema da escalabilidade, onde, por meio da adição de nós em determinados estágios (blocos funcionais) foi possível dividir os dados e realizar o processamento em paralelo, o que reduziu o tempo e permitiu que o requisito do tempo de processamento pudesse ser atingido de forma satisfatória.

A versão atual apresenta como principal limitação a não existência de uma ferramenta visual, seja para a configuração, ou mesmo a implementação ou implantação dos módulos nos nós computacionais do arranjo. Isso faz com que existam um conjunto de arquivos texto (XML) de configuração, que em alguns casos devem ser definidos de forma manual e em cada um dos nós, o que faz com que este possa ser um ponto de erro, ou mesmo, de perda de desempenho, pela má definição destes parâmetros.

Outra questão importante quanto ao uso da plataforma CoP-WS está ligada com a sua filosofia de trabalho. Diferente de outras tecnologias onde é possível fazer uma implementação incremental, ou seja, faz-se modelos pequenos e vai agregando novas funcionalidades, a plataforma CoP-WS apresenta essa capacidade apenas quando considerado o arranjo, onde novas funcionalidades podem ser inseridas em nós e estes adicionados no arranjo, e o fluxo de processamento modificado para sua inclusão. Mas se os nós não forem bem projetados, estes devem ser refeitos por completo, pois estes podem ser considerados como sendo pontos de baixo desempenho, ou de produção de dados duvidosos. Este tema faz com que a plataforma CoP-WS seja muito dependente de seu projeto, e das definições do fluxograma deste.

5.1 Trabalhos Futuros

Na seção anterior foi apresentada uma discussão sobre a plataforma CoP-WS, sobre os resultados obtidos, tanto pelo uso da plataforma, durante sua implementação, quanto pelos resultados obtidos pelas aplicações implementadas como prova de viabilidade desta. Nesta seção é apresentada uma lista com a relação de trabalhos futuros para dar continuidade à plataforma CoP-WS, seguinte:

- Implementar uma configuração de rede dedicada que possa atender as necessidades particulares da arquitetura CoP-WS.
- Implementar um Nó Computacional Genérico, que possa ser utilizado em qualquer bloco funcional configurado. Este Nó iniciaria o processamento desabilitado, mas poderia vir entrar em funcionamento substituindo qualquer nó computacional do arranjo, devido a falha ou mesmo aumentar o poder de processamento de um bloco funcional devido a um aumento na carga de processamento.
- Criação de uma interface gráfica para configuração dos nós do arranjo - A versão atual do CoP-WS não possui quaisquer interfaces gráficas que facilitem o processo de configuração e acompanhamento do processamento de aplicações implementadas e que estejam sendo processadas com a utilização desta tecnologia;
- Criação de uma ferramenta para configuração e distribuição da carga de processamento nos nós do arranjo - seria uma ferramenta que permitiria o desenvolvimento de um fluxograma, e do cadastro das especificações do problema a ser processado. Com base nestas informações, e por meio de simulações, ou por uma base de conhecimento, esta ferramenta poderia fazer a determinação da quantidade dos nós necessária por cada um dos

blocos funcionais;

- Utilização da teoria dos grafos para a criação de uma ferramenta de apoio no cálculo de rotas da plataforma;
- Incluir a FPGA como dispositivo de coprocessamento;
- Realizar a implementação de dois, ou mais aplicativos, utilizando nós do arranjo de forma simultânea;
- Desenvolvimento de uma linguagem de programação em blocos onde cada bloco seja um bloco funcional existente no arranjo computacional;
- Implementar uma interface de aquisição de dados para o arranjo interferométrico do BDA, de forma a realizar o processamento de dados reais e em um ambiente de produção.
- A inclusão de servidores UDDI para a publicação e automatização dos serviços oferecidos pela rede.

REFERÊNCIAS

ABERGER, C. *SOAP and .net Services for Apache HTTP Server*. 2010. Disponível em: <http://alien.cern.ch/cache/gsoap2.7-2.7.10/gsoap/mod_gsoap/mod_gsoap-0.6/apache_13/apache_index.html>.

ACCELLERA. *EDA Industry Working Groups*. 2006. Disponível em: <http://www.vhdl.org>.

ALMASI, S. G.; GOTTLIEB, A. Highly parallel computing. In: *The Benjamin Cummings Publishing Company Inc.* [S.l.: s.n.], 1994.

ALTAMIMI, Z.; COLLILIEUX, X.; MÉTIVIER, L. Itrf2008: an improved solution of the international terrestrial reference frame. *Journal of Geodesy*, Springer-Verlag, v. 85, n. 8, p. 457–473, 2011. ISSN 0949-7714. Disponível em: <http://dx.doi.org/10.1007/s00190-011-0444-4>.

AMAMIYA, M. et al. Research on programming languages for massively parallel processing. In: *International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN)*. [S.l.: s.n.], 1994. p. 443–450.

APACHE. *The Apache Software Foudation*. 2013. Disponível em: <http://www.apache.org>.

APACHE. *HDFS Architecture Guide*. 2013. Disponível em: <http://hadoop.apache.org/docs/stable/hdfs_design.html>.

APACHE. *Welcome to Apache Hadoop*. 2013. Disponível em: <http://hadoop.apache.org/>.

ARECIBO. *National Astronomy and Ionosphere Center Arecibo Observatory*. 2011. Disponível em: <http://www.naic.edu/index_scientific.php>.

ASANOVIC, K. et al. A view of the parallel computing landscape. *Commun. ACM*, ACM, New York, NY, USA, v. 52, n. 10, p. 56–67, out. 2009. ISSN 0001-0782. Disponível em: <http://doi.acm.org/10.1145/1562764.1562783>.

AVISON, A.; GEORGE, S. J. A graphical tool for demostrating the techniques of radio interferometry. *European Journal of Physics*, n. 34, p. 7–17, 2013.

AYGUADE, E. et al. An extension of the starss programming model for platforms with multiple gpus. Springer-Verlag, Berlin, Heidelberg, p. 851–862, 2009.

BARFORD, T. *Map-Reduce on Mongo*. May 2010. Disponível em: <http://tarnbarford.net/journal/mapreduce-on-mongo>.

BARROSO, L. A.; HÖLZLE, U. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. [S.l.]: Morgan & Claypool, 2009.

BLUMOFE, R. Cilk: an efficient multithreaded runtime system. *Journal of Parallel and Distributed Computing*, v. 37, n. 1, p. 55–69, 1996.

BOOTH, D. et al. *Web Service Architecture*. February 2004. Disponível em: <http://www.naic.edu/index_scientific.php>.

BUSTAMAM, A.; BURRAGE, K.; HAMILTON, N. Fast parallel markov clustering in bioinformatics using massively parallel computing on gpu with cuda and ellpack-r sparse format. *Computational Biology and Bioinformatics, IEEE/ACM Transactions on*, v. 9, n. 3, p. 679–692, 2012. ISSN 1545-5963.

CARRIERO, N.; GELERNTER, D. How to write parallel programs: a guide to the perplexed. In: . New York, NY, USA: ACM, 1989. v. 21, n. 3, p. 323–357. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/72551.72553>.

CECATTO, J. R. Capítulo 4 o sol. In: DIVISÃO DE ASTROFÍSICA. *Curso de Introdução à Astronomia e Astrofísica - Capítulo 4 O Sol*. [S.l.]: Instituto Nacional de Pesquisas Espaciais, 2003.

CECATTO, J. R. *Interferometria e Síntese de Abertura*. [S.l.], 2011.

CENTURION, A. M. Análise de desempenho de algoritmos paralelos utilizando plataforma de portabilidade. In: *Dissertação de Mestrado, Instituro de Ciências Matemáticas e de Computação da Universidade de São Paulo (ICMC/USP).* [S.l.: s.n.], 1998.

CHAPMAN, B.; JOST, G.; PAS, R. V. D. *Using OpenMP: Portable Shared Memory Parallel Programming (Scientific and Engineering Compution)*. [S.l.]: Cambridge, MIT Press, 2007.

CILK 5.4.6 - Reference Manual. [S.l.]: Supercomputing Technologies Group MIT Laboratory for Computer Science, 2009.

CLUSTER, S. *GPU Programming*. 2013. Disponível em: <http://selkie.macalester.edu/csinparallel/modules/GPUProgramming/build/html/Introduction/Introduction.html>.

DELLER, A. et al. Difx: A software correlator for very long baseline interferometry using multiprocessor computing environments. In: *Astronomical Society of the Pacific*. [S.l.: s.n.], 2007. v. 119, p. 318–336.

DELLER, A. T. et al. Difx-2: A more flexible, efficient, robust, and powerful software correlator. *The Astronomical Society of the Pacific*, v. 123, n. 901, p. 275–287, March 2011.

EGELEN, R. A. van. *gSOAP 2.8.15 User Guide*. [S.l.]: GENIVIA INV, 2013.

FARIA, C. *Uma nova Estratégia de Otimização de Arranjos Interferométricos Aplicada ao Brazilian Decimetric Array*. Tese (Doutorado) — INPE, São José dos Campos - Brasil, 2006.

FERNANDES, F. C. R. *Espectrógrafo digital decimétrico de band larga e investigação de "FLARES"solares em ondas decimétricas e radios-X*. [S.l.]: INPE - Instituto Nacional de Pesquisas Espaciais, 1997.

FLYNN, J. M. Some computer organizations and their effectiveness. In: *IEEE TOC*. [S.l.: s.n.], 1972. p. 948–960.

GOKHALE, M.; GRAHAM, P. *Reconfigurable computing: Accelerating Computation with Field-Programmable Gate Arrays*. [S.l.]: Netherlands: Springer, 2005.

GONZALEZ, J. G. The offline software of the pierre auger observatory: Lessons learned. *IEEE International Symposium of Parallel and Distributer Processing with Applications*, n. 10th, p. 557–564, 2012.

GORDER, P. F. Coming soon: Research in a cloud. *Computing in Science & Engineering*, v. 10, n. 6, p. 6–10, 2008.

GSCHWIND, M. et al. A novel simd architecture for the cell heterogeneous chip-multiprocessor. In: *HOT CHIPS*. [S.l.: s.n.], 2005. v. 17.

HARALICK, R. M.; SHANMUGAN, K. Textural features for image classification. *IEEE Transactions on Systems, Man, and Cybernetics. SMC*, p. 610–621, 1973.

HARRIS, C. J. *A Parallel Model for the Heterogeneous Computation of Radio Astronomy Signal Correlation*. Tese (Doutorado) — The University of Western Australia, July 2009.

HENRY, C.; PETERS, J. *Near set Evaluation And Recognition (NEAR) System*. January 2012. Disponível em: <http://wren.ece.umanitoba.ca/index.php?option=com_content&view=article&id=119&Itemid=93>.

HENRY, C. J. *Near Sets: Theory and Applications*. Tese (Doutorado) — University of Manitoba, September 2010.

HUNG, C.-L. et al. Efficient gpgpu-based parallel packet classification. *Trust, Security and Privacy in Computing and Communications*, v. 10, p. 1367–1374, 2011.

HWANG, K.; BRIGGS, A. F. Computer architecture and parallel processing. In: *McGraw-Hill*. [S.l.: s.n.], 1984.

HWANG, K.; XU, Z. Scalable parallel computing. In: *WCB/McGraw-Hill.* [S.l.: s.n.], 1998.

HWU, W. mei W.; KEUTZER, K.; MATTSON, T. The concurrency challenge. In: *IEEE Design and Test of Computers*. [S.l.: s.n.], 2008. v. 25, n. 4, p. 312–320.

INTEL. *Intel Many Integrated Core Architecture (Intel MIC Architecture) – Advanced*. 2013. Disponível em: <http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html>.

JAMSEK, D.; VANHENSBERGEN, E. *Experiences with Hybrid Clusters*. [S.l.]: Cluster Computing and Workshops, 2009. CLUSTER '09. IEEE International Conference on, 2009.

JONES, D. H. et al. *GPU versus FPGA for High Productivity Computing*. [S.l.]: Field Programmable Logic and Applications (FPL), 2010 International Conference on, 2010.

KAUSHIK, R. T.; BHANDARKAR, M.; NAHRSTEDT, K. Evaluation and analysis of greenhdfs: A self-adaptive, energy-conserving variant of the hadoop distributed file system. *International Conference on Cloud Computing Technology and Science*, v. 2nd, p. 274–287, 2010.

KIRNER, C. Arquiteturas de sistemas avançados de computação. In: *Anais da Jornada EPUSP/IEEE em Sistemas de Computação de Alto Desempenho*. [S.l.: s.n.], 1991. p. 307–353.

LABARTA, J. Starss: a programming model for the multicore era. In: *In PRACE Workshop 'New Languages & Future Technology Prototypes' at the Leibniz Supercomputing Centre in Garching (Germany)*. [S.l.: s.n.], 2010.

LEE, Y. Web services registry implementation for processing quality of service. *International Conference on Advanced Language Processing and Web Information Technology*, p. 538–543, 2008.

LIANG, T.-Y. et al. A cloud computing service for fast audio source signal separation. In: *Machine Learning for Signal Processing (MLSP), 2011 IEEE International Workshop on*. [S.l.: s.n.], 2011. p. 1–6.

LIMA, J. V. F. et al. Exploiting concurrent gpu operations for efficient work stealing on multi-gpus. *International Symposium on Computer Architecture and High Performance Computing*, Columbia University, New York, USA, 2012.

MARQUES, J. P. *Reconhecimento de Padrões*. [S.l.], 2000. Disponível em: <http://paginas.fe.up.pt/ jmsa/recpad/>.

MARS, J.; TANG, L.; HUNDT, R. Heterogeneity in "homogeneous"warehouse-scale computers: A performance opportunity. *Computer Architecture Letters*, v. 10, n. 2, p. 29–32, 2011.

MCCREARY, C.; GILL, H. Automatic determination of grain size for efficient parallel processing. In: *Comunication of the ACM New York*. [S.l.: s.n.], 1989. v. 32, n. 9, p. 1073–1078.

MENTEC, F. L.; GAUTIER, T.; DANJEAN, V. *The X-Kaapi's Application Programming Interface. Part I: Data Flow Programming*. [S.l.], dez. 2011. 48 p. Disponível em: <http://hal.inria.fr/hal-00648245>.

NAKAJIMA, H. et al. The nobeyama radioheliograph. *Proceedings of the IEEE*, v. 92, n. 5, p. 705 – 713, May 1994.

NAVAUX, A. P. Introdução do processamento paralelo. In: *RBC - Revista Brasileira de Computação*. [S.l.: s.n.], 1989. p. 31–43.

NOBEYAMA. *Nobeyama Radioheliograph Event List*. 2013. Disponível em: <http://solar.nro.nao.ac.jp/norh/html/event/>.

NRO. *Nobeyama Radioheliograph*. 2012. Disponível em: <http://solar.nro.nao.ac.jp/norh/html/introduction.html>.

NVIDIA. *CUDA C Programming Guide*. [S.l.]: NVIDIA, 2012.

ORLOWSKA, E. *Semantics of vague concepts, Applications of Rough Sets*. [S.l.]: IPI PAN, 1982. (Prace IPI PAN, 469).

PAPAZOGLOU, M. P. *Web Services: Principles & Technology*. 3th. ed. Edinburgh Gate, Harlowm England: Pearson Education, 2008.

PAWLAK, Z. Rough set theory and its applications to data analysis. *Cybernetics and Systems*, v. 29, n. 7, p. 661–688, 2002.

PENG, J.; CHEN, H.; SHI, S. The gpu-based string matching system in advanced ac algorithm. *Computer and Information Technology (CIT)*, v. 10, p. 1158–1163, 2010.

PETERS, J. F.; WASILEWSKI, P. Foundations of near sets. *Inf. Sci.*, Elsevier Science Inc., New York, NY, USA, v. 179, n. 18, p. 3091–3109, ago. 2009. ISSN 0020-0255. Disponível em: <http://dx.doi.org/10.1016/j.ins.2009.04.018>.

PIAZZESI, R. et al. Algorithm for real time flare detection. *Memorie delta Supplementi*, v. 19, n. 109, 2012.

PINTO, V. G. *Ambientes de Prorgamação Paralela Híbrida*. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul, 2011.

PLANET, A. *The Arecibo Observatory of the SETI Project*. March 2010. Disponível em: <http://www.amusingplanet.com/2010/03/arecibo-observatory-of-seti-project.html>.

POLI, G. et al. *Voice Command Recognition with Dynamic Time Warping (DTW) using Graphics Processing Units (GPU) with Compute Unified Device Architecture (CUDA)*. São José dos Campos - Brasil: 19th Iternational Symposium on Computer Architecture and Hight Performance Computing, 2007. 19-25 p.

POLI, G.; SAITO, J. H.; CECATTO, J. R. Mapreduce and object-oriented paragihm applied to the radio interferometer signal correlation in gpu environment. *IADIS International Conference Applied Computing*, Madrid - Spain, v. 19th, October 2012.

QU, M. et al. Automated recognition of solar flares in real-time data. *Wireless and Optical Communications Conference*, n. 14, p. 102–106, April 2005.

QUINN, J. M. Designing efficient algorithms for parallel computers. In: *McGraw Hill.* [S.l.: s.n.], 1987.

REINDERS, J. *Intel threading building blocks: outfitting C++ for multi-core processor parallelism.* [S.l.]: O'Reilly, 2007.

ROSHI, D. A. *Low Frequency Radio Astronomy - Chapter 9 - Correlator - II: Implementation*. [S.l.]: GMRT, 2003.

ROSHI, D. A. Correlator - ii: Implementation. In: ____. [S.l.]: GMRT - Giant Metrewave Radio Telescope, 2009. cap. Chapter 9.

Rygl, K. L. J. et al. Parallaxes and proper motions of interstellar masers toward the cygnus x star-forming complex. *A&A*, v. 539, p. A79, 2012. Disponível em: <http://dx.doi.org/10.1051/0004-6361/201118211>.

SALNIKOV, A. et al. Moldyngrid virtual laboratory as a part of ukrainian academic grid infrastructure. In: *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, 2009. IDAACS 2009.* [S.l.: s.n.], 2009. p. 237–240.

SAULE, E.; CATALYUREK, U. An early evaluation of the scalability of graph algorithms on the intel mic architecture. In: *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International*. [S.l.: s.n.], 2012. p. 1629–1639.

SAVYTSKI, O. V. et al. Integrated tools for molecular dynamics simulation data analysis in the moldyngrid virtual laboratory. *Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*, v. 6, p. 208–211, 2011.

SEONG, B.; AHN, J.; SUNG, W. Parallel implementation of an error diffusion halftoning algorithm with a general purpose graphics processing unit. In: *IEEE International Conference on Image Processing (ICIP), 2010 17th*. [S.l.: s.n.], 2010. p. 3777–3780. ISSN 1522-4880.

SHAN, A. Heterogeneous processing: a strategy for augmenting moore's low. *Linux Journal*, n. 142, p. 32–40, 2006.

SHIH, F. Y. Automated detection and classification of solar image features. *Handbook of Pattern Recognition and Computer Vision*, World Scientific, London, 2010.

SI, X. et al. Parallel optimization of queries in xml dataset using gpu. *Parallel Architectures, Algorithms and Programming (PAAP)*, v. 4, p. 190–194, 2011.

SOUZA, M. A. Avaliação de rotinas de comunicação ponto-a-ponto do mpi. In: *Dissertação de Mestrado, Instituto de Ciências Matemáticas de São Carlos (ICMSC), Universidade de São Paulo (USP).* [S.l.: s.n.], 1996.

SUBRAMANIAN, K. R. *Report of the Scientific Work Done During the Period June 12, 2011 to June 11,2012*. INPE - São José dos Campos, 2012.

SUNDERAM, V. S. et al. The pvm concurrent computing system: evolution experiences and threads. In: *Parallel Computing*. [S.l.: s.n.], 1994. p. 531–545.

THOMAS, J. P.; THOMAS, M.; GHINEA, G. Modeling of web services flow. *International Conference on Adavanced Language Processing and Web Information Technology*, p. 391–398, 2003.

TSOI, K. H.; LUK, W. Axel: A heterogeneous cluster with fpgas and gpus. *FPGA*, New York, NY, USA, p. 115–124, 2010.

VERILOG. *Verilog Resources*. 2012. Disponível em: <http://www.verilog.com>.

W3C. *Web Services Description Language (WSDL) 1.1*. 2001. Disponível em: <http://www.w3.org/TR/wsdl>.

W3C. *DTD Tutorial*. 2007. Disponível em: <http://www.w3schools.com/dtd/>.

W3C. *Introduction to XML*. 2007. Disponível em: <http://www.w3schools.com/xml/xml_whatis.asp>.

W3C. *SOAP Version 1.2 Part 1: Messaging Framework (Second Edition)*. 2007. Disponível em: <http://www.w3.org/TR/soap12-part1/>.

W3C. *SOAP Tutorial*. [S.l.], 2008.

W3C. *WSDL Tutorial*. [S.l.], 2008.

W3C. *W3C - HTML*. 01 2013. Disponível em: <http://www.w3.org/html/>.

WAGNER, J.; RITAKARI, J. Difx - a distributed fx-style software correlation. In: *The Publications of the Astronomical Society of the Pacific*. [S.l.]: Metsähovi Radio Observatory, 2009. v. 119, n. 853, p. 318–336.

WILSON, T. L.; ROHLFS, K.; HÜTTEMEISTER, S. *Tools of radio astronomy.* 5th. ed. [S.l.]: Springer, 2009. (Astronomy & Astrophysics).

WOODS, A. *Accelerating Software Radio Astronomy FX Correlation with GPU and FPGA Co-processors*. Tese (Doutorado) — University of Cape Town, October 2010.

WU, E.; LIU, Y. Emerging technology about gpgpu. *Circuits and Systems. APCCAS IEEE Asia Pacific Conference*, p. 618–622, 2008.

ZHANG, J.; LANG, B.; DUAN, Y. An xml data placement strategy for distributed xml storage and parallel query. *International Conference on Parallel and Distributed Computing, Applications and Technologies*, n. 12th, p. 433– 439, 2011.

Apendice A

Arquivos de Configuração dos Serviços Web Service

Listagem do código XML do protocolo WSDL da transformada de Hilbert.

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <definitions name="nodeHilbert"
4   targetNamespace="urn:nodeHilbert"
5   xmlns:tns="urn:nodeHilbert"
6   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
7   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
8   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
10  xmlns:ns="urn:nodeHilbert"
11  xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
12  xmlns:HTTP="http://schemas.xmlsoap.org/wsdl/http/"
13  xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
14  xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
15  xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
16  xmlns="http://schemas.xmlsoap.org/wsdl/">
17
18 <types>
19
20  <schema targetNamespace="urn:nodeHilbert"
21    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
22    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
23    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
24    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
25    xmlns:ns="urn:nodeHilbert"
```

```
26    xmlns="http://www.w3.org/2001/XMLSchema"
27    elementFormDefault="unqualified"
28    attributeFormDefault="unqualified">
29    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
30    <!-- operation request element -->
31    <element name="nodeHilbert">
32     <complexType>
33      <sequence>
34       <element name="time-cycle" type="xsd:dateTime" minOccurs="1" maxOccurs
          ="1"/><!-- ns__nodeHilbert::time_cycle -->
35       <element name="total" type="xsd:int" minOccurs="1" maxOccurs="1"/><!--
           ns__nodeHilbert::total -->
36       <element name="index" type="xsd:int" minOccurs="1" maxOccurs="1"/><!--
           ns__nodeHilbert::index -->
37      </sequence>
38     </complexType>
39    </element>
40   </schema>
41
42  </types>
43
44  <message name="nodeHilbert">
45   <part name="Body" element="ns:nodeHilbert"/><!--
       ns__nodeHilbert::ns__nodeHilbert -->
46  </message>
47
48  <portType name="nodeHilbertPortType">
49   <operation name="nodeHilbert">
50    <documentation>Service definition of function ns__nodeHilbert</
        documentation>
51    <input message="tns:nodeHilbert"/>
52   </operation>
53  </portType>
54
55  <binding name="nodeHilbert" type="tns:nodeHilbertPortType">
56   <SOAP:binding style="document" transport="http://schemas.xmlsoap.org/soap/
       http"/>
57   <operation name="nodeHilbert">
58    <SOAP:operation soapAction=""/>
59    <input>
60       <SOAP:body parts="Body" use="literal"/>
61    </input>
62    <output>
```

```
63        <SOAP:body parts="Body" use="literal"/>
64      </output>
65     </operation>
66    </binding>
67
68    <service name="nodeHilbert">
69     <documentation>gSOAP 2.8.16 generated service definition</documentation>
70     <port name="nodeHilbert" binding="tns:nodeHilbert">
71      <SOAP:address location="200.18.98.156/nodeHilbert.cgi"/>
72     </port>
73    </service>
74
75    </definitions>
```

Listagem do código XML do protocolo WSDL do FFT

```
1
2   <?xml version="1.0" encoding="UTF-8"?>
3   <definitions name="nodeFFT"
4    targetNamespace="urn:nodeFFT"
5    xmlns:tns="urn:nodeFFT"
6    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
7    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
8    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
10   xmlns:ns="urn:nodeFFT"
11   xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
12   xmlns:HTTP="http://schemas.xmlsoap.org/wsdl/http/"
13   xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
14   xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
15   xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
16   xmlns="http://schemas.xmlsoap.org/wsdl/">
17
18  <types>
19
20   <schema targetNamespace="urn:nodeFFT"
21    xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
22    xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
23    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
24    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
25    xmlns:ns="urn:nodeFFT"
26    xmlns="http://www.w3.org/2001/XMLSchema"
27    elementFormDefault="unqualified"
```

```
28    attributeFormDefault="unqualified">
29    <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
30    <!-- operation request element -->
31    <element name="nodeFFT">
32     <complexType>
33      <sequence>
34       <element name="time-cycle" type="xsd:dateTime" minOccurs="1" maxOccurs
          ="1"/><!-- ns__nodeFFT::time_cycle -->
35       <element name="total" type="xsd:int" minOccurs="1" maxOccurs="1"/><!--
           ns__nodeFFT::total -->
36       <element name="index" type="xsd:int" minOccurs="1" maxOccurs="1"/><!--
           ns__nodeFFT::index -->
37      </sequence>
38     </complexType>
39    </element>
40   </schema>
41
42  </types>
43
44  <message name="nodeFFT">
45   <part name="Body" element="ns:nodeFFT"/><!-- ns__nodeFFT::ns__nodeFFT -->
46  </message>
47
48  <portType name="nodeFFTPortType">
49   <operation name="nodeFFT">
50    <documentation>Service definition of function ns__nodeFFT</documentation>
51    <input message="tns:nodeFFT"/>
52   </operation>
53  </portType>
54
55  <binding name="nodeFFT" type="tns:nodeFFTPortType">
56   <SOAP:binding style="document" transport="http://schemas.xmlsoap.org/soap/
       http"/>
57   <operation name="nodeFFT">
58    <SOAP:operation soapAction=""/>
59    <input>
60       <SOAP:body parts="Body" use="literal"/>
61    </input>
62    <output>
63       <SOAP:body parts="Body" use="literal"/>
64    </output>
65   </operation>
66  </binding>
```

```
67
68 <service name="nodeFFT">
69  <documentation>gSOAP 2.8.16 generated service definition</documentation>
70  <port name="nodeFFT" binding="tns:nodeFFT">
71   <SOAP:address location="200.18.98.67/nodeFFT.cgi"/>
72  </port>
73 </service>
74
75 </definitions>
```

Listagem do código XML do protocolo WSDL da Multiplicação Cruzada

```
1
2 <?xml version="1.0" encoding="UTF-8"?>
3 <definitions name="nodeMultiplication"
4  targetNamespace="urn:nodeMultiplication"
5  xmlns:tns="urn:nodeMultiplication"
6  xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
7  xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
8  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
10  xmlns:ns="urn:nodeMultiplication"
11  xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
12  xmlns:HTTP="http://schemas.xmlsoap.org/wsdl/http/"
13  xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
14  xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
15  xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
16  xmlns="http://schemas.xmlsoap.org/wsdl/">
17
18 <types>
19
20  <schema targetNamespace="urn:nodeMultiplication"
21   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
22   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
23   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
24   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
25   xmlns:ns="urn:nodeMultiplication"
26   xmlns="http://www.w3.org/2001/XMLSchema"
27   elementFormDefault="unqualified"
28   attributeFormDefault="unqualified">
29   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
30   <!-- operation request element -->
31   <element name="nodeMultiplication">
```

```
32      <complexType>
33       <sequence>
34        <element name="time-cycle" type="xsd:dateTime" minOccurs="1" maxOccurs
              ="1"/><!-- ns__nodeMultiplication::time_cycle -->
35        <element name="total" type="xsd:int" minOccurs="1" maxOccurs="1"/><!--
              ns__nodeMultiplication::total -->
36        <element name="index" type="xsd:int" minOccurs="1" maxOccurs="1"/><!--
              ns__nodeMultiplication::index -->
37       </sequence>
38      </complexType>
39     </element>
40    </schema>
41
42  </types>
43
44  <message name="nodeMultiplication">
45   <part name="Body" element="ns:nodeMultiplication"/><!--
        ns__nodeMultiplication::ns__nodeMultiplication -->
46  </message>
47
48  <portType name="nodeMultiplicationPortType">
49   <operation name="nodeMultiplication">
50    <documentation>Service definition of function ns__nodeMultiplication</
         documentation>
51    <input message="tns:nodeMultiplication"/>
52   </operation>
53  </portType>
54
55  <binding name="nodeMultiplication" type="tns:nodeMultiplicationPortType">
56   <SOAP:binding style="document" transport="http://schemas.xmlsoap.org/soap/
        http"/>
57   <operation name="nodeMultiplication">
58    <SOAP:operation soapAction=""/>
59    <input>
60       <SOAP:body parts="Body" use="literal"/>
61    </input>
62    <output>
63       <SOAP:body parts="Body" use="literal"/>
64    </output>
65   </operation>
66  </binding>
67
68  <service name="nodeMultiplication">
```

```
69  <documentation>gSOAP 2.8.16 generated service definition</documentation>
70  <port name="nodeMultiplication" binding="tns:nodeMultiplication">
71   <SOAP:address location="200.18.98.154/nodeMultiplication.cgi"/>
72  </port>
73 </service>
74
75 </definitions>
```

Listagem do código XML do protocolo WSDL do Integração

```
1
2  <?xml version="1.0" encoding="UTF-8"?>
3  <definitions name="nodeIntegration"
4   targetNamespace="urn:nodeIntegration"
5   xmlns:tns="urn:nodeIntegration"
6   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
7   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
8   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
9   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
10  xmlns:ns="urn:nodeIntegration"
11  xmlns:SOAP="http://schemas.xmlsoap.org/wsdl/soap/"
12  xmlns:HTTP="http://schemas.xmlsoap.org/wsdl/http/"
13  xmlns:MIME="http://schemas.xmlsoap.org/wsdl/mime/"
14  xmlns:DIME="http://schemas.xmlsoap.org/ws/2002/04/dime/wsdl/"
15  xmlns:WSDL="http://schemas.xmlsoap.org/wsdl/"
16  xmlns="http://schemas.xmlsoap.org/wsdl/">
17
18 <types>
19
20  <schema targetNamespace="urn:nodeIntegration"
21   xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
22   xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/"
23   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
24   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
25   xmlns:ns="urn:nodeIntegration"
26   xmlns="http://www.w3.org/2001/XMLSchema"
27   elementFormDefault="unqualified"
28   attributeFormDefault="unqualified">
29   <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
30   <!-- operation request element -->
31   <element name="nodeIntegration">
32    <complexType>
33     <sequence>
```

```
34      <element name="time-cycle" type="xsd:dateTime" minOccurs="1" maxOccurs
            ="1"/><!-- ns__nodeIntegration::time_cycle -->
35      <element name="total" type="xsd:int" minOccurs="1" maxOccurs="1"/><!--
            ns__nodeIntegration::total -->
36      <element name="index" type="xsd:int" minOccurs="1" maxOccurs="1"/><!--
            ns__nodeIntegration::index -->
37     </sequence>
38    </complexType>
39   </element>
40  </schema>
41
42 </types>
43
44 <message name="nodeIntegration">
45  <part name="Body" element="ns:nodeIntegration"/><!--
        ns__nodeIntegration::ns__nodeIntegration -->
46 </message>
47
48 <portType name="nodeIntegrationPortType">
49  <operation name="nodeIntegration">
50   <documentation>Service definition of function ns__nodeIntegration</
        documentation>
51   <input message="tns:nodeIntegration"/>
52  </operation>
53 </portType>
54
55 <binding name="nodeIntegration" type="tns:nodeIntegrationPortType">
56  <SOAP:binding style="document" transport="http://schemas.xmlsoap.org/soap/
        http"/>
57  <operation name="nodeIntegration">
58   <SOAP:operation soapAction=""/>
59   <input>
60      <SOAP:body parts="Body" use="literal"/>
61   </input>
62   <output>
63      <SOAP:body parts="Body" use="literal"/>
64   </output>
65  </operation>
66 </binding>
67
68 <service name="nodeIntegration">
69  <documentation>gSOAP 2.8.16 generated service definition</documentation>
70  <port name="nodeIntegration" binding="tns:nodeIntegration">
```

```
71    <SOAP:address location="200.18.98.151/nodeIntegration.cgi"/>
72   </port>
73  </service>
74
75  </definitions>
```