LEONARDO DE OLIVEIRA SANTOS

Caracterização de tarefas usando Redes Neurais e CUDA

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

São José do Rio Preto
2015

LEONARDO DE OLIVEIRA SANTOS

Caracterização de tarefas usando Redes Neurais e CUDA

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

Orientador:
Prof. Dr. Aleardo Manacero Jr.

São José do Rio Preto
2015

LEONARDO DE OLIVEIRA SANTOS

Caracterização de tarefas usando Redes Neurais e CUDA

Monografia apresentada ao Departamento de Ciências de Computação e Estatística do Instituto de Biociências, Letras e Ciências Exatas da Universidade Estadual Paulista "Júlio de Mesquita Filho", como parte dos requisitos necessários para aprovação na disciplina Projeto Final.

Prof. Dr. Aleardo Manacero Jr.

Leonardo de Oliveira Santos

Banca Examinadora:
Prof. Dr. Leandro Alves Neves
Prof.ª Dr.ª Renata Spolon Lobato

São José do Rio Preto
2015

À minha família

AGRADECIMENTOS

Primeiramente agradeço a Deus por sua presença em minha vida e sua constante proteção.

Agradeço aos meus pais por estarem sempre no meu pé, me cobrando, me empurrando pra frente, pois acreditam no meu potencial. Agradeço por terem me criado da forma que fizeram e por toda educação que recebi. Se hoje posso dizer que possuo caráter, eu devo isso a vocês.

À minha irmã por ter enfrentado tanta coisa ao meu lado, pelas conversas que nós tivemos, por todas as brincadeiras e momentos divertidos que passamos.

Agradeço aos meus tios por todos os momentos em que pude contar com vocês para desabafar, pelos momentos em que vocês puderam me aconselhar, por estarem presentes para me ajudar sempre que precisei.

Aos meus avós por serem os melhores avós do mundo, por sempre se doarem quando veem que alguém precisa, por todas as vezes que me levaram para Rio Preto, pelos momentos de descontração que passamos juntos.

Agradeço à Liliane e ao Carlos por momentos extraordinários em que pude relaxar, esquecer um pouco do estresse do cotidiano e me acolherem como um filho. Agradeço ao Matheus por todos os momentos de descontração e brincadeiras, pela companhia...

Ao meu amor, Talita, por ter ficado ao meu lado apesar de tudo que enfrentamos, por ter me motivado e me dado forças pra continuar quando tudo que eu queria fazer era desistir. Obrigado por me fazer feliz, por me ajudar a ser uma pessoa melhor e por sempre estar ao meu lado quando eu precisei, apesar da distância.

Ao Prof. Dr. Aleardo Manacero Jr. por toda a paciência que teve comigo nos últimos anos, por nunca ter desistido de mim e acreditado na minha capacidade para realizar esse trabalho. Por ter se tornado mais que um professor para mim, mas alguém em que eu posso buscar ajuda, tirar dúvidas e sempre tomar um cafezinho na cantina.

À Prof.ª Dr.ª Renata Spolon Lobato por ter me convidado para participar do GSPD, ter me ajudado sempre que pode e pelas ótimas conversas na hora do café. Obrigado por acreditar em mim e ser a ótima professora que é.

A todos os meus amigos e colegas da faculdade, por todo o conhecimento que pude receber e compartilhar com vocês. Obrigado Gabriel, Willian, Heitor, Danilo, Rafael, Leandro, Diogo, Denison, Mario, Brunno, Thiago, Igor, Victor Hugo, Cassio... Todos vocês tiveram um papel essencial em algum momento da minha vida acadêmica e sou grato por cada desses momentos.

“Everything not saved will be lost”
- Nintendo “Quit Screen” message

RESUMO

Ao trabalhar com simulação de grades de computadores nos deparamos com a necessidade de casar corretamente a carga de trabalho aplicada ao tipo de sistema que se pretende avaliar. Isso ocorre por que usuários diferentes podem ter interesse em tarefas com diferentes características, dependendo do objetivo de suas aplicações. Essa carga de trabalho pode ser originada a partir de arquivos de traces, que armazenam o histórico de execução de um sistema real, permitindo que os resultados das simulações se aproximem mais de ambientes reais. Assim, é importante que os arquivos de traces a serem usados tenham o perfil de tarefas adequado às necessidade do usuário. Para tanto é preciso que ele seja caracterizado a fim de obter um aproveitamento melhor de suas peculiaridades em simulações. Uma forma de fazer tal caracterização é através do uso de redes neurais, em que as tarefas de um arquivo são analisadas para indicar se têm um perfil mais próximo de computação intensiva ou de comunicação intensiva. Como arquivos de traces podem conter milhões de tarefas, é interessante encontrar formas de acelerar essa caracterização, o que pode ser feito através de paralelismo, obtido, por exemplo, por GPUs. Nesse projeto se avaliou o uso de GPUs para a implementação de uma rede neural de Kohonen, paralelizada em CUDA. O desempenho dessa rede foi comparado com uma implementação sequencial, com resultados que permitem indicar que a paralelização é vantajosa em determinadas situações.

ABSTRACT

While simulating computer grids one finds the need for a correct match between the workload applied to the type of the system under evaluation. This occurs because distinct users may be interested in tasks with different characteristics, depending on the goals of their applications. This workload can be originated from trace files, which store the execution history of a real system, allowing that the from the simulations become closer to actual environments. Therefore, it is important that the trace files to be used have the same tasks' profile that is suited to the user's needs. To do so it is necessary to characterize the file in order to achieve a better employment of its peculiarities in the simulations. One possible approach for characterization is to use neural networks, where the tasks in the trace are evaluated to identify if they are cpu-bounded or communication-bounded. Since trace files can contain millions of tasks, it is interesting to find approaches to speedup the characterization process, what can be done through parallelism, using a GPU, for example. In this work we evaluated the use of GPUs to implement a Kohonen neural network, parallelized with CUDA. The performance of such network was compared with a sequential version, presenting results that indicate that the parallelization is more advantageous in some situations.

ÍNDICE

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SOM: *Self-Organizing Maps*
GSPD: Grupo de Sistemas Paralelos e Distribuídos
GP-GPU: *General-Purpose Computing on Graphics Processing Units*
GPU: *Graphics Processing Unit*
CPU: *Central Processing Unit*
RNA: Rede Neural Artificial
RNK: Rede Neural de Kohonen
CUDA: *Compute Unified Device Architecture*
ULA: Unidade Lógica Aritmética
SIMD: *Single Instruction, Multiple Data*

Capítulo 1

Introdução

1.1 – Motivação

Atualmente a necessidade por cada vez mais poder computacional tornou-se uma constante em diversos setores de produção, desde pesquisas científicas ao setor industrial. A fim de suprir essa carência, foram desenvolvidas tecnologias que fossem capazes de realizar a tarefas de alto custo computacional em um tempo factível e com alto desempenho.

O melhor caminho para obter alto desempenho é com computação paralela (Branco, 2004), que se tornou possível devido à alta conectividade dos recursos computacionais proporcionada pela internet e as grandes redes de computadores existentes hoje. Nesse contexto, a utilização de grades computacionais, que consistem em vários computadores compartilhando seus recursos para a execução de uma única tarefa paralelamente, se tornou uma alternativa interessante.

As tarefas que serão executadas por uma grade computacional formam aquilo que chamamos de carga de trabalho (Feitelson, 2011). O perfil dessas cargas de trabalho devem ser compatíveis com o da grade para conseguir um uso eficiente do sistema, já que cada conjunto de tarefas e cada grade possuem características próprias.

Para permitir uma análise desse desempenho são criados registros da execução das tarefas numa grade, chamados de arquivos de *traces*, que armazenam dados importantes das tarefas e podem ser usados para simular a mesma execução em outras grades de computadores ou até mesmo em simuladores.

O Grupo de Sistemas Paralelos e Distribuídos (GSPD) criou um simulador de grades de computadores chamado iSPD (Aleardo, et al, 2012) que pode usar arquivos de *traces* em suas simulações e também gera seus próprios arquivos *traces*.

Partindo do princípio de que caracterizando as tarefas contidas em cada um desses arquivos de *traces* conseguiremos obter um desempenho melhor na simulação, pois usaremos um conjunto de tarefas compatível com o sistema simulado, procuramos uma forma de fazer isso.

A forma escolhida para realizar tal caracterização foi o uso de Redes Neurais Artificiais (RN), que são sistemas que buscam utilizar alguns princípios de organização e aprendizagem do cérebro humano para fazer com que o computador, ao receber determinados estímulos, analise-os e gere uma resposta coerente baseado nesses estímulos. Nesse trabalho, serão usadas Redes Neurais de Kohonen (Kohonen, 1989), também conhecidas como SOM - *Self-Organizing Maps* (Mapas Auto-Organizáveis). Redes Neurais são utilizadas para realizar reconhecimento de padrões, controle motor, entre outras aplicações e o processamento delas pode ser paralelizado. O processo de paralelização pode ser feito utilizando *General-Purpose Computing on Graphics Processing Units* (GP-GPU), ou seja, usar a *Graphics Processing Unit* (GPU) para executar as tarefas que seriam executadas pela CPU.

1.2 – Objetivos

O objetivo desse trabalho é criar duas versões de uma Rede Neural de Kohonen (RNK) para realizar a caracterização de *traces*, sendo que uma delas realizará todo o processamento sequencialmente na CPU e a outra realizará o mesmo processamento na GPU utilizando a tecnologia CUDA da NVIDIA, e a comparação do desempenho das duas ferramentas. Essa comparação será feita submetendo as duas versões da RNK aos mesmos arquivos de *traces* e comparando a velocidade de processamento e a precisão da caracterização entre elas.

1.3 – Organização

Esse trabalho será organizado em quatro capítulos, além desta introdução. No capítulo 2 serão discutidos brevemente os conceitos de Cargas de Trabalho, Redes Neurais e Redes Neurais de Kohonen, a plataforma CUDA (*Compute Unified Device Architecture*), a organização do hardware em que ela é utilizada. No capítulo 3 será apresentada a implementação das RNKs e o algoritmo de treinamento utilizado. No capítulo 4 veremos os testes realizados com as RNKs e seus resultados. Finalmente, no capítulo 5 serão apresentadas as conclusões do trabalho realizado.

Capítulo 2

Fundamentação Teórica

Nesse capítulo serão apresentados os conceitos básicos de Redes Neurais, Redes Neurais de Kohonen, cargas de trabalho e da plataforma CUDA.

2.1 Redes Neurais

A criação das redes neurais artificiais foi motivada a partir do momento em que foi percebido que o cérebro humano processa suas informações de maneira completamente diferente do computador digital. Segundo (Haykin, 1998), o cérebro humano é um computador altamente complexo, não linear e paralelo que possui a capacidade de organizar seus elementos estruturais, os neurônios, para realizar processos como reconhecimento de padrões, percepção, controle motor, entre outros.

Um ponto interessante é o processo de aprendizagem dos neurônios. No inicio do seu desenvolvimento, os neurônios possuem certa maleabilidade e conseguem adaptar-se ao ambiente em que estão e aprendem a processar as informações que recebem com mais eficácia com o passar do tempo. Então, baseando-se na estrutura do cérebro humano e em seus neurônios, foram criadas as Redes Neurais Artificiais (RNA), que podem ser definidas da seguinte forma, segundo (Haykin, 1998):

"A rede neural é um processador distribuído e massivamente paralelo formado por unidades de processamento simples, que tem a propensão de armazenar conhecimento exponencial e torna-lo disponível para uso. Ela lembra o cérebro em dois aspectos:

1. *Conhecimento é adquirido pela rede neural do seu ambiente através do processo de aprendizagem.*
2. *A conexão entre neurônios, conhecida como pesos sinápticos, se fortalece e são usados para adquirir conhecimento."*

Existem diferentes tipos de arquiteturas de redes neurais e elas estão diretamente relacionadas com o algoritmo de treinamento da RNA. Os três tipos de arquitetura existentes são as redes de Camada Simples Acíclica, de Camada Múltipla Acíclica e as redes Recorrentes ou Cíclicas, que serão descritas a seguir.

2.1.1 Rede de Camada Simples Acíclica

É a forma mais simples de rede particionada em camadas. Ela possui uma camada de entrada com nós que transmitem informações aos nós da camada de saída, mas o processo contrário não acontece (Figura 2.1).

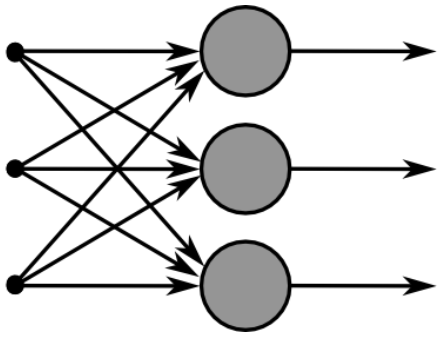

Figura 2.1: Rede de Camada Simples Acíclica

2.1.2 Rede de Múltiplas Camadas Acíclica

Essa arquitetura de RNA é parecida com a anterior, mas possui mais camadas entre as camadas de entrada e saída da rede. Essas camadas servem para intervir entre as duas camadas principais de alguma forma útil. Elas podem extrair determinadas estatísticas valiosas das entradas e adicionar peso aos dados de entrada de acordo com a relevância de cada um deles. Essas camadas são chamadas de “camadas escondidas” ou “*hidden layers*” (Figura 2.2).

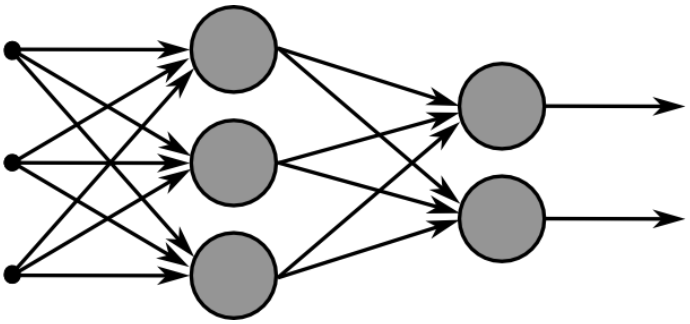

Figura 2.2: Rede de Múltiplas Camadas Acíclica

2.1.3 Redes Recorrentes ou Cíclicas

A diferença entre as redes recorrentes e as redes acíclicas é que elas possuem no mínimo um *loop* de realimentação, ou seja, pelo menos um nó da camada de saída da RNA enviará seus resultados para um nó da camada de entrada da rede. Esse tipo de rede pode ou não possuir camadas escondidas e também existe a possibilidade de um nó de saída usar seus resultados para se realimentar (*self-feedback)* (Figura 2.3).

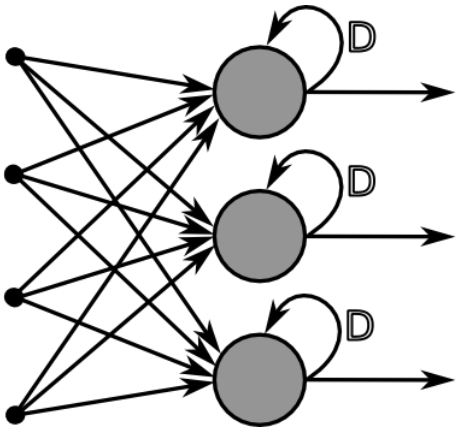

Figura 2.3: Rede recorrente ou cíclica com *self-feedback*

Redes Neurais Artificiais também podem ser classificadas de acordo com seu método de aprendizagem e existem diversos modos de realizar essa tarefa, mas nesse trabalho apresentaremos apenas dois desses métodos, sendo eles o método Supervisionado e o Não Supervisionado.

Em RNAs com aprendizado supervisionado podemos imaginar que existe um professor que mostra à RNA qual é a saída esperada quando se recebe determinada entrada. Ela então ajusta o peso de seus neurônios a fim de produzir a saída desejada. Para fazer uso desse paradigma de aprendizado é necessário o uso de um número grande de dados de entradas com suas respectivas saídas esperadas para serem usados na fase de treinamento da rede. Dessa forma, é como se o conhecimento do professor fosse transferido para a RNA até que esta esteja pronta para realizar seu trabalho sem supervisão.

Já em RNAs com aprendizado não supervisionado, como o nome sugere, não há um "professor" para ensinar a rede que saída produzir caso receba determinada entrada de dados. Portanto, a rede recebe os dados de entrada e ajusta o peso de seus neurônios de acordo um algoritmo competitivo baseado em uma regra de aprendizagem e o neurônio vencedor é quem responderá por aquele tipo de entrada. A estratégia "*winner-takes-all*" (vencedor leva tudo) é a mais simples dessas regras.

2.2 Redes Neurais de Kohonen

Assim como as demais RNAs, as Redes Neurais de Kohonen (RNK) são formadas por neurônios conectados entre si, formando uma rede. Ela é uma rede de camada simples acíclica, com uma camada de entrada e uma camada competitiva (Figura 2.4). Ela utiliza um paradigma de aprendizado não supervisionado específico chamado de Self-Organizing Maps (SOM) ou Mapas Auto Organizáveis, e algoritmo de aprendizado competitivo. Esse paradigma de aprendizagem analisa os sinais de entrada e os organiza em grupos de acordo com a semelhança entre os sinais.

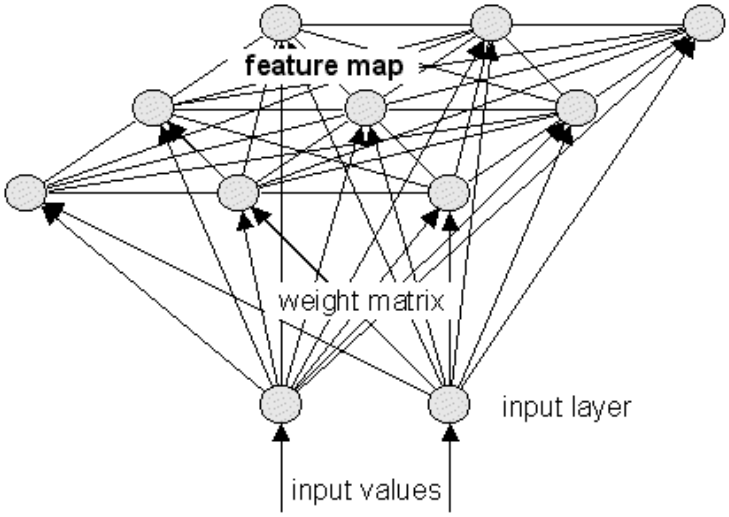

Figura 2.4: Rede Neural de Kohonen (NNWJ, 2004)

Esse tipo de rede neural é vantajoso para esse trabalho por que além de possuir as facilidades conceituais de redes neurais para reconhecimento de padrões, ele usa uma estrutura paralela de seus neurônios. Sua estrutura básica é separada em duas camadas, sendo uma delas a de entrada e a outra, a camada competitiva, ou camada de processamento, na qual todos os nós são conectados dentro de cada camada e entre as duas camadas.

A fase de treinamento é separada em etapas, sendo elas a de entrada de dados, de competição, de cooperação e adaptação sináptica. Na etapa de entrada de dados, um vetor de entrada é apresentado à primeira camada da RNK, ela então disponibiliza esses valores à segunda camada. Aí começa a etapa competitiva, em que os neurônios da segunda camada competem para encontrar um vencedor seguindo o algoritmo que será apresentado mais a frente. Após encontrar um vencedor, inicia a fase de cooperação, quando é definida uma vizinhança do neurônio vencedor. Finalmente, os pesos do neurônio vencedor e sua vizinhança são ajustados para que a resposta do neurônio vencedor seja mais específica na próxima iteração do algoritmo. Esse processo se repete até o fim da fase de treinamento.

O algoritmo de treinamento apresentado aqui foi extraído de (Sewo, 2003). Assumindo que o padrão de entrada seja:

$$\boldsymbol{x} = [x_1, x_2, \dots, x_n]$$

Os pesos de cada neurônio sejam:

$$w_j = \left[w_{j1}, w_{j2}, \dots, w_{jn}\right], \qquad j = 1, 2, \dots, n$$

O primeiro passo do algoritmo é determinar o grau de similaridade entre o vetor de entrada e cada neurônio da RNK. Isso é feito calculando a distância Euclidiana entre os vetores x e w_j. Ela é calculada utilizando a fórmula:

$$\sqrt{\sum_{i=1}^{n}(x_i - w_{ji})^2}$$

na qual i é o índice de cada posição do vetor de entrada e do neurônio e j é o índice do neurônio do qual deseja saber a similaridade do vetor de entrada.

O neurônio com maior similaridade, ou seja, com a menor distância vence a competição. Inicia-se então a fase de cooperação. A vizinhança, denotada por $V_{k(x)}(t)$, é formada por neurônios próximos ao vencedor e diminui a cada iteração (Figura 2.5). Os pesos então são atualizados para todos os neurônios da vizinhança usando a seguinte condição:

$$w_j(t+1) = \begin{cases} w_j(t) + \eta(t)\left[x(t) - w_j(t)\right], & j \epsilon V_{k(x)}(t) \\ w_j(t), & caso\ contrário \end{cases}$$

na qual $\eta(t)$ é o parâmetro de taxa de aprendizado e $V_{k(x)}(t)$ é a função de vizinhança centrada em torno do neurônio vencedor k(x); ambos variam dinamicamente durante o processo de aprendizagem para obter melhores resultados.

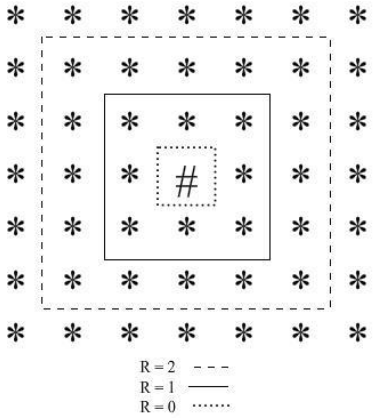

Figura 2.5: Vizinhança do neurônio vencedor alterada com o tempo

Dessa forma, os neurônios afetados por tal ajuste se tornam mais parecidos com o padrão de entrada apresentado. Assim, o neurônio w_j torna-se mais propício a vencer a competição quando é apresentado como entrada o vetor *x* ou uma entrada semelhante a ele. O valor de $\eta(t)$ deve ser inicializado em um valor entre zero e um e então ser decrescido conforme as iterações do treino ocorrem. Uma taxa de decrescimento aceitável é:

$$\eta = \eta_o \left(1 - \frac{t}{T}\right)$$

na qual *t* é a iteração de treino atual e T é o número total de iterações de treino a ser feita, assim η começa com um valor η_0 e diminui com o tempo, mas ficando sempre acima de 0.

Já o tamanho da vizinhança começa com uma largura relativamente grande e diminui depois de muitas iterações, de acordo com:

$$d = d_o \left(1 - \frac{t}{T}\right)$$

na qual d é a distância da i-ésima unidade até a borda da vizinhança.

2.3 CUDA

CUDA (*Compute Unified Device Architecture*) é uma arquitetura criada pela NVIDIA (NVIDIA, 2014) que foi implantada em suas placas de vídeo com o objetivo de aliviar várias limitações existentes em processadores gráficos mais antigos que não permitiam que eles fossem usados para *General Purpose Computing*. Ao contrário de outras arquiteturas existentes, a arquitetura CUDA permite que cada unidade lógica aritmética (ULA) seja usada por um programa que deseja realizar *general purpose computing*. Para isso, as ULAs foram construídas de acordo com os padrões IEEE para aritmética de pontos flutuantes de precisão única (*single-precision floating-point arithmetic*) e não especificamente para operações gráficas.

Mas mesmo com essas modificações em suas placas de vídeo, não havia uma maneira prática de utilizar tais recursos sem ter que disfarçar seus programas de operações gráficas. Foi então que a NVIDIA adicionou alguns comandos à linguagem C para facilitar o acesso a tais recursos de suas placas gráficas e tornou público o compilador para a linguagem CUDA C. Desde então a linguagem CUDA C vem sendo utilizada para resolver problemas das mais diversas naturezas como acelerar o processamento de imagens médicas, com dinâmica de fluidos, ciência ambientais, entre muitas outras aplicações.

A principal diferença entre a execução de processos na CPU e executá-los na GPU é que a CPU também é responsável por várias outras funções do dispositivo como, por exemplo, o controle do sistema operacional, entre outras operações fundamentais para o funcionamento do dispositivo. Já a GPU, além de não precisar realizar tais funções, possui seu foco nas ULAs, portanto acelerando o processamento de certas tarefas.

A fim de facilitar a programação, a arquitetura CUDA organiza seus *threads* em blocos e esses blocos, em *grids* (Figura 2.6). Esses conjuntos devem ser chamados explicitamente no programa para serem utilizados. A execução segue o modelo *Single Instruction, Multiple Data* (SIMD), fazendo com que a mesma instrução seja enviada a todas as ULAs e sejam executadas com dados diferentes (NVIDIA, 2013).

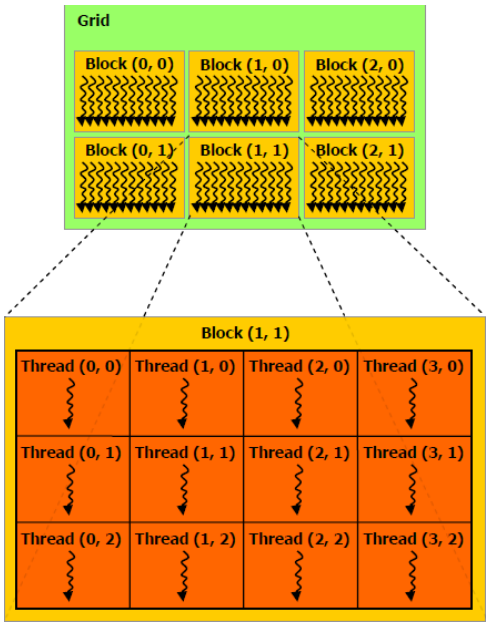

Figura 2.6: Organização de *Grids*, Blocos e *Threads* (NVIDIA, 2014)

Para que os dados sejam utilizados na GPU, eles devem ser copiados para a sua memória global explicitamente. Os threads acessarão os dados que estiverem lá, mas para realizarem compartilhamento de dados entre eles é necessário utilizar a memória compartilhada, pois a memória global possui alta latência, portanto não é recomendável utiliza-la para isso. A única limitação da memória compartilhada é que ela é acessível apenas para os *threads* do mesmo bloco (Figura 2.7).

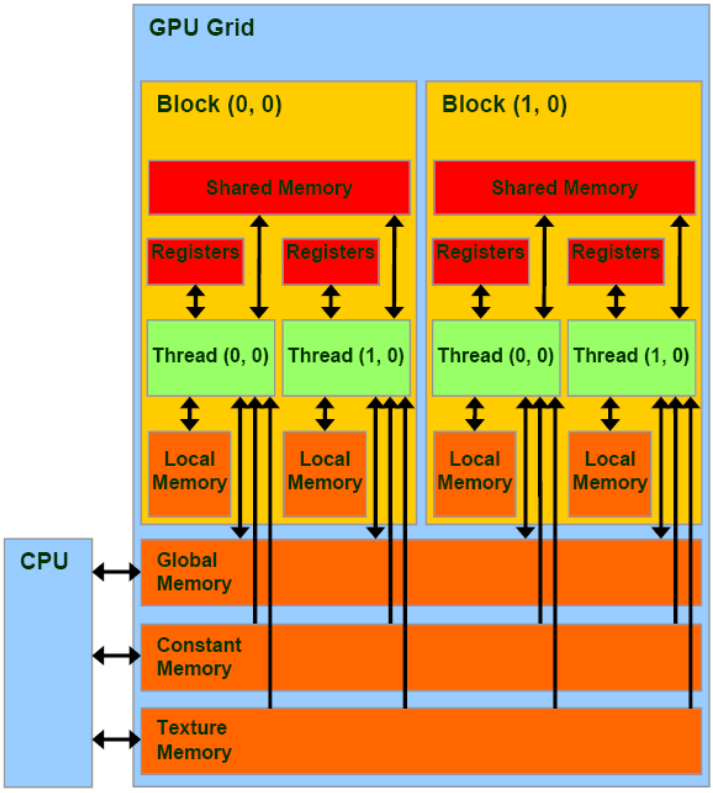

Figura 2.7: Modelo de memória CUDA (NVIDIA, 2014)

Por fim, o gerenciamento da memória da GPU também deve ser realizado explicitamente, o que significa que a alocação da memória, cópia de dados tanto da CPU para a GPU quanto da GPU para a CPU e o uso das memórias compartilhadas devem ser feitos de forma explícita no programa.

2.4 Cargas de Trabalho

A avaliação de desempenho é um elemento básico da ciência da computação, sendo usada para criar novos componentes, ajustar parâmetros, entre outras funções. Mas avaliar o desempenho de um sistema sem um conjunto de testes adequado para aquele sistema pode levar à conclusões erradas e prejudicar o uso dos recursos computacionais. Segundo Feitelson (Feitelson, 2011), os três principais fatores que influenciam no desempenho de um sistema computacional são: o design do sistema, sua implementação e a carga de trabalho que o sistema será submetido.

As cargas de trabalho a qual o sistema será submetido são de extrema importância, pois se não utilizarmos cargas condizentes ao sistema, obteremos resultados equivocados. Por isso, é importante que se busquem modelos de cargas adequados ao sistema que está sendo estudado.

Um arquivo de *trace* contém registros de valores que caracterizam a execução de uma instância de determinada aplicação, ou conjunto de aplicações, em um determinado ambiente, por meio de sensores e rotinas de monitoramento (Silva, 2012). *Traces* gerados por grades de computadores reais podem ser salvos e usados em simuladores de grades, como o iSPD (Manacero, et al, 2010), simulador criado no Grupo de Sistemas Paralelo e Distribuídos (GSPD).

A seguir temos um exemplo de arquivo de trace gerado pelo iSPD. Nas tarefas 1 e 2, temos exemplos de “Computação Intensiva”, as tarefas 3 e 4 representam tarefas de “Comunicação Intensiva” e finalmente, as tarefas 5 e 6 representam tarefas “Neutras”. O valor “cpsz” indica quanto tempo a tarefa passou sendo executada na CPU e o valor “cmsz” indica quantos Mbits de dados foram transferidos durante a execução. Em função de a simulação ter sido feita em uma rede com 100Mbps de largura de banda, para obter o tempo gasto transferindo dados, dividimos o valor de “cmsz” por 100, assim obtendo valores que podem ser comparados entre si.

$<?xml\ version = "1.0"\ encoding = "ISO-8859-1"\ standalone = "no"?>$
$<!DOCTYPE\ system\ SYSTEM\ "iSPDcarga.dtd">$
$<system>$
$<trace>$
$<format\ kind = "SWF" />$
$<task\ id = "1"\ arr = "0"\ sts = "1"\ cpsz = "1451"\ cmsz = "97"\ usr = "user1" />$
$<task\ id = "2"\ arr = "1460"\ sts = "1"\ cpsz = "3726 "\ cmsz = "139"\ usr = "user1" />$
$<task\ id = "3"\ arr = "5198"\ sts = "1"\ cpsz = "93"\ cmsz = "37843"\ usr = "user1" />$
$<task\ id = "4"\ arr = "6269"\ sts = "1"\ cpsz = "87"\ cmsz = "24157"\ usr = "user2" />$
$<task\ id = "5"$ arr= "1728" $sts = "1"\ cpsz = "2927"$ cmsz= "25418" $usr = "user1" />$
$<task\ id = "6"\ arr = "2020"\ sts = "1"\ cpsz = "2731"\ cmsz = "17731"\ usr = "user3" />$

A fim de tornar o uso das cargas de trabalho mais eficiente, tornou-se necessária a caracterização delas, possibilitando a escolha do tipo de carga de trabalho que seria utilizada para realizar simulações e diminuindo o desperdício de recursos. Em (Jardim, 2004) a caracterização de cargas de trabalho em sistemas paralelos é separada em

etapas. Dentre elas, a mais crítica é a escolha dos parâmetros usados para a classificação, que pode ser feito em função de sua dependência ou independência da arquitetura do sistema, analisando o comportamento do sistema ao lidar com tais cargas.

Já no trabalho de (Calzarossa, Serazzi, 1993), são citados passos necessários para a caracterização de qualquer tipo de cargas de trabalhos que são:

- Escolher parâmetros que são capazes de descrever o comportamento das cargas de trabalho;
- O uso de ferramentas adequadas para a análise e monitoramento de desempenho;
- Coleção de experimentos com suas devidas medições;
- Análise dos dados das cargas de trabalho;
- Construção de modelos estáticos/dinâmicos das cargas de trabalho.

Encontramos também em (Elnaffar, Martin, 2006) que as técnicas de caracterização de cargas de trabalho podem ser classificadas como estáticas e dinâmicas. As técnicas estáticas utilizam as características intrínsecas da carga de trabalho, como a correlação entre elas, classes de transação, entre outros elementos que não são alterados com o tempo. Já as técnicas dinâmicas fazem uso de gráficos de comportamento, métodos de regressão, entre outros métodos focados em como a carga de trabalho se comporta com o tempo. Às vezes essas técnicas usadas separadamente não disponibilizam resultados satisfatórios, dependendo da necessidade existente. Por isso, não é incomum encontrarmos técnicas que fazem uso de elementos estáticos e dinâmicos.

Dentre as técnicas estáticas podemos citar:

- **Estatísticas Descritivas:** são usadas técnicas para identificar as propriedades estáticas da carga de trabalho, podendo encontrar a média, dispersão, variância, entre outras;
- **Histogramas de Parâmetro Único:** são uma representação visual da variação de determinados parâmetros em intervalos pré-determinados.
- **Histogramas de Parâmetros Múltiplos:** são iguais aos de parâmetro único, mas com a capacidade de expressar a correlação entre parâmetros.

Dentre as técnicas dinâmicas podemos citar:

- **Modelos de Markov:** Esse modelo é representado por uma matriz de transição, que nos dá a probabilidade de passar para um próximo estado, dado o estado atual. Um diagrama de transição de estado pode ser criado a partir dessa matriz.
- **Previsões usando Redes Neurais:** Apesar da dificuldade de conseguir previsões perfeitas, as redes neurais podem ser utilizadas para obter bons resultados na caracterização de cargas de trabalho.

2.5 Aplicações de GPU em Redes Neurais

GPUs com CUDA tem sido usadas para as mais diversas aplicações devido à sua facilidade de programação e uso. Elas são usadas em áreas da medicina para realizar o mapeamento 3D de órgãos, na geologia para acelerar o processamento de imagens e em várias outras áreas de acordo com suas necessidades. Como Redes Neurais podem se beneficiar com a paralelização de seu processamento, CUDA se tornou uma ferramenta vantajosa nessa área também.

(Bako, et al, 2012) implementa uma Rede Neural Spiking para classificar imagens de eletroencefalogramas de acordo com certas categorias apresentadas em seu treinamento e utiliza CUDA nessa RN para conseguir um desempenho melhor. No trabalho de (Uetz, 2009), é apresentada uma solução para o reconhecimento de objetos de larga escala usando uma rede neural com processamento paralelizado usando CUDA, conseguindo alcançar uma velocidade de processamento 82 vezes mais rápida que uma CPU com um núcleo. Em (Pendlebury, et al, 2012) é realizada uma simulação de uma rede neural paralelizada em CUDA para executar o programa “Mine Hunter”, na qual é obtido um desempenho 80% melhor do que a execução em um processador Intel i5-2500 de 3.3GHz.

Capítulo 3

Implementação e algoritmo

Nesse capítulo serão apresentadas a organização das RNKs implementadas assim como seus componentes e os algoritmos usados para realizar o treinamento das RNKs. Foram implementadas duas RNKs, sendo que uma delas realiza a caracterização dos *traces* sequencialmente, usando a linguagem C, e a outra paralelamente utilizando a tecnologia CUDA com a linguagem CUDA C.

3.1. Especificação das RNKs

Foi proposta neste trabalho a implementação de RNKs para realizar a caracterização de arquivos de *traces* gerados pelo simulador de grades de computadores iSPD. Para realizar essa caracterização, foi implementada uma RNK que percorre sequencialmente o arquivo de *trace* e, a fim de acelerar o processo de caracterização, foi implementada uma RNK que realiza a mesma tarefa paralelamente utilizando a tecnologia CUDA.

Com isso, foi necessário primeiramente definir o número de características que poderiam ser extraídas dos arquivos que seriam caracterizados. Após o estudo dos *traces* gerados pelo iSPD, foi definido que os dados relevantes para caracterização contido neles eram o *Communication Size* (cmsz) e o *Computational Size* (cpsz), que definem qual a quantidade de dados que são transferidos na comunicação entre os nós da grade e a quantidade processada nos nós, respectivamente. Com isso, cada registro de tarefa contido no *trace* seria classificado como “Comunicação Intensiva”, “Computação Intensiva” ou “Neutra”.

Após definidos os possíveis tipos de caracterização, foi necessário definir o número de neurônios que as RNKs possuiriam. Devido ao fato de não haver uma quantidade grande de parâmetros a serem analisados, foi definido que seria utilizado um neurônio para cada tipo de caracterização, portanto três neurônios. As RNKs foram estruturadas em duas camadas, sendo uma delas a camada de entrada e a outra a camada competitiva, onde são realizadas as fases de treinamento e a caracterização das tarefas (Figura 3.1). Foram utilizadas duas camadas em função do número de parâmetros utilizados e por não haver a necessidade de aplicar pesos a eles após a fase de treinamento e de caracterização.

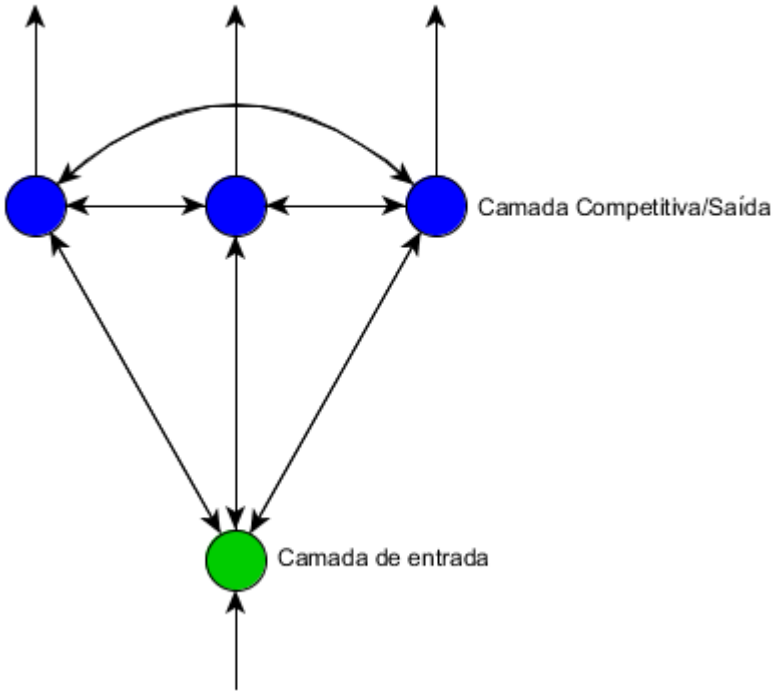

Figura 3.1: Estrutura da Rede Neural de Kohonen implementada.

3.2. Implementação das RNKs

Após definir a estrutura das RNKs, foi iniciada a fase de implementação delas. Primeiramente foi criada uma estrutura de dados chamada "neurônio" que possui três variáveis que armazenam a identificação, a porcentagem de tempo de processamento e a porcentagem de tempo de comunicação de cada um. Essas variáveis serão utilizadas na fase de treinamento e de classificação seguindo o algoritmo citado na sessão 2.2 para calcular a distância Euclidiana entre os dados de entrada e os neurônios e encontrar o vencedor. Elas são inicializadas com valores similares aos que definiriam cada característica apresentada no tópico anterior.

Depois de preparar os neurônios para a fase de treinamento, foi necessário definir duas estruturadas de dados: a estrutura "*trace",* que armazena a identificação de cada tarefa, o seu tempo de chegada, seu estado ao final da execução, carga de comunicação, carga de processamento e a identificação do usuário que solicitou a execução da tarefa, e a estrutura "porcentagem", que armazena a identificação de cada tarefa, a porcentagem de tempo que ela executando e a porcentagem de tempo que essa tarefa passou se comunicando com outros nós da grade. Essas estruturas foram utilizadas para realizar a entrada de dados dos *traces* para a RNK.

A função que faz a entrada dos dados lê o arquivo de *trace* e armazena os dados de cada tarefa nas estruturas "*trace"*. Para isso, é calculado o número de linhas do arquivo e um vetor dessa estrutura é alocado dinamicamente com o número de posições necessárias. Após essa análise, é calculado o tempo que cada tarefa passou sendo

processada e quanto tempo ela levou para realizar transferência de dados com os outros nós e esses dados são armazenados em um vetor da estrutura "porcentagem".

Com isso, pode ser iniciada a etapa de treinamento dos neurônios. Esses neurônios foram inicializados, como citado anteriormente, por que dessa forma a convergência desses valores na fase de treinamento seria acelerada. Depois de feitos os ajustes necessários, inicia-se a fase competitiva, em que é calculado para cada tarefa a distância Euclidiana entre ela e cada um dos neurônios. A distância é calculada seguindo a seguinte fórmula:

$$dist = \sqrt{\begin{matrix}\left(porc\_comm_{tarefa_j} - porc\_comm_{neurônio_i}\right)^2 + \\ \left(porc\_comp_{tarefa_j} - porc\_comp_{neurônio_i}\right)^2\end{matrix}} \quad (1)$$

na qual trace$_j$. $porc_{comm}$ e $trace_j.porc_{comp}$ são os valores da porcentagem do tempo de comunicação e porcentagem do tempo de processamento da tarefa que será classificada e $neuronio_i.porc_{comm}$ e $neuronio_i.porc_{comp}$ representam os mesmos valores, mas do neurônio que se deseja calcular a distância. Então, o neurônio que possui a menor distância do vetor de entrada, é o vencedor da fase competitiva.

A fase cooperativa teve que ser modificada nesse trabalho em função do número de neurônios na rede. Por termos um número pequeno de neurônios, o raio de vizinhança para o reajuste dos valores foi desconsiderada, fazendo com que apenas o neurônio vencedor tenha os valores de suas variáveis atualizados. A atualização foi feita seguindo a seguinte fórmula:

$$porc\_comm_{neurônio_i} = porc\_comm_{neurônio_i} + \eta * \left(porc\_comm_{tarefa_j} - porc\_comm_{neurônio_i}\right) \quad (2)$$

$$porc\_comp_{neurônio_i} = porc\_comp_{neurônio_i} + \eta * \left(porc\_comp_{tarefa_j} - porc\_comp_{neurônio_i}\right) \quad (3)$$

e a taxa de aprendizagem η é atualizada utilizada a função decrescente:

$$\eta = \eta * \frac{\left(1 - e^{-\eta * k}\right)}{\left(1 + e^{-\eta * k}\right)} \quad (4)$$

na qual k é uma constante utilizada para regular a taxa de decrescimento da variável η. Na implementação das RNKs, ela foi inicializada com valor 12, por que utilizando esse valor, o peso dos neurônios convergia quando η valia 0,0001. Foi usada essa função, pois ela oferece uma variação da taxa de aprendizagem interessante para esse trabalho. Com isso, é finalizada a etapa de treinamento dos neurônios. Até esse ponto, não há diferença entre as duas RNKs, pois o treinamento deve ser feito sequencialmente. O processamento das duas nessa etapa é feito pela CPU. Após a fase de treinamento, é criado um log com os valores das variáveis de cada neurônio que é lido toda vez que a RNK é executada para que o treinamento não precise ser executado em toda

inicialização. A seguir serão apresentadas as diferenças na implementação das duas RNKs.

3.2.1. RNK Sequencial

Após o treinamento, os neurônios estão prontos para realizar a caracterização, e essa etapa do processamento das RNKs é feita de forma diferente para a que foi implementada sequencialmente usando a linguagem C e a implementada em CUDA C.

Na RNK escrita em C, é criado um loop onde o usuário digita o nome do arquivo que deve ser caracterizado. Então é feito o processamento do arquivo da mesma forma que era feito na etapa de treinamento: o arquivo de *trace* é analisado e seus dados são importados para a estrutura de dados "*trace*", são calculadas as porcentagens de tempo de processamento e comunicação, depois armazena-se esses dados na estrutura de dados "porcentagens". Em seguida é utilizada a mesma função do cálculo da distância Euclidiana para encontrar o neurônio vencedor, portanto o mais similar à tarefa que está sendo caracterizada. Conhecendo o neurônio que mais se assemelha à tarefa atual, a RKN afirma que ela possui a mesma característica do neurônio vencedor. Isso é repetido para cada linha do arquivo sequencialmente pela CPU (Figura 3.2).

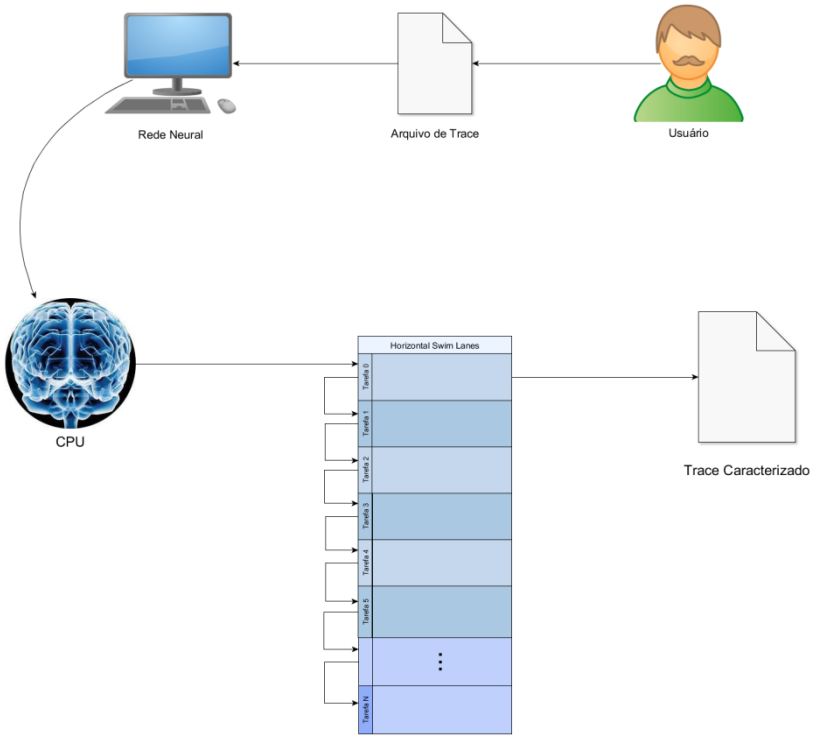

Figura 3.2: Diagrama da execução da RNK Sequencial

Após percorrer cada tarefa contida no arquivo de *trace*, é calculada a porcentagem de tarefas que contém cada característica e o arquivo será caracterizado de acordo com o traço que mais se evidenciou nas tarefas. Depois de finalizada a caracterização, o usuário tem a opção de selecionar outro arquivo para inserir na RNK ou finalizar o programa.

3.2.2. RNK Paralela

Na RNK escrita em CUDA C, foram necessárias algumas alterações para que a caracterização fosse feita paralelamente. Para isso, foram utilizados alguns elementos característicos da linguagem CUDA C, diferentes da linguagem C tradicional.

Algumas diferenças ficam mais evidentes no primeiro contato com um programa escrito em CUDA C. A primeira é a chamada "*__global__*" antes do nome de uma função que deve ser executada pela GPU. Isso é feito justamente para diferenciar funções executadas pelo processador das funções executadas na GPU. Essas funções, mesmo não sendo executadas pela CPU, podem ser chamadas tanto por ela quanto por outra função executada na GPU. Outra chamada possível é a "__device__", que caracteriza funções que são chamadas exclusivamente por funções executadas na GPU.

Na RNK implementada em CUDA, fazemos a chamada da função que realiza a classificação dos traces na GPU da seguinte forma:

<<< *grid_size*, *block_size* >>> nome_funcao(parametro1, …)

na qual *grid_size* é o número de blocos que serão utilizados e *block_size* é o número de *threads* em cada bloco.

Então, após chamar a função que será executada na GPU com o número de blocos e *threads* que desejamos, devemos designar tarefas para cada *thread* classificar. Isso é feito usando a variável "*tid*" que é o identificador de cada *thread* que executará a função. Ele é um valor único calculado utilizando o identificador do bloco que o *thread* está localizado (*blockIdx.x*), a dimensão dos blocos (*blockDim.x*) e a identificação do *thread* no bloco (*threadIdx.x*).

Com esses valores, cada *thread* é colocado em um loop que fará com que eles calculem a distância e encontrem o neurônio vencedor para cada tarefa com identificador múltiplo do valor número de total de *threads* somado ao identificador de cada *thread* chamado. Dessa forma, todos os *threads* de cada bloco continuam trabalhando até que não haja mais tarefas a serem caracterizadas, independente do tamanho do arquivo (Figura 3.3).

Após a caracterização de cada tarefa é realizada a caracterização do arquivo de *trace*. É calculada a porcentagem de tarefas que contém cada característica e o arquivo será caracterizado de acordo com o traço que mais se evidenciou nas tarefas. Depois de finalizada a caracterização, o usuário tem a opção de selecionar outro arquivo para inserir na RNK ou finalizar o programa.

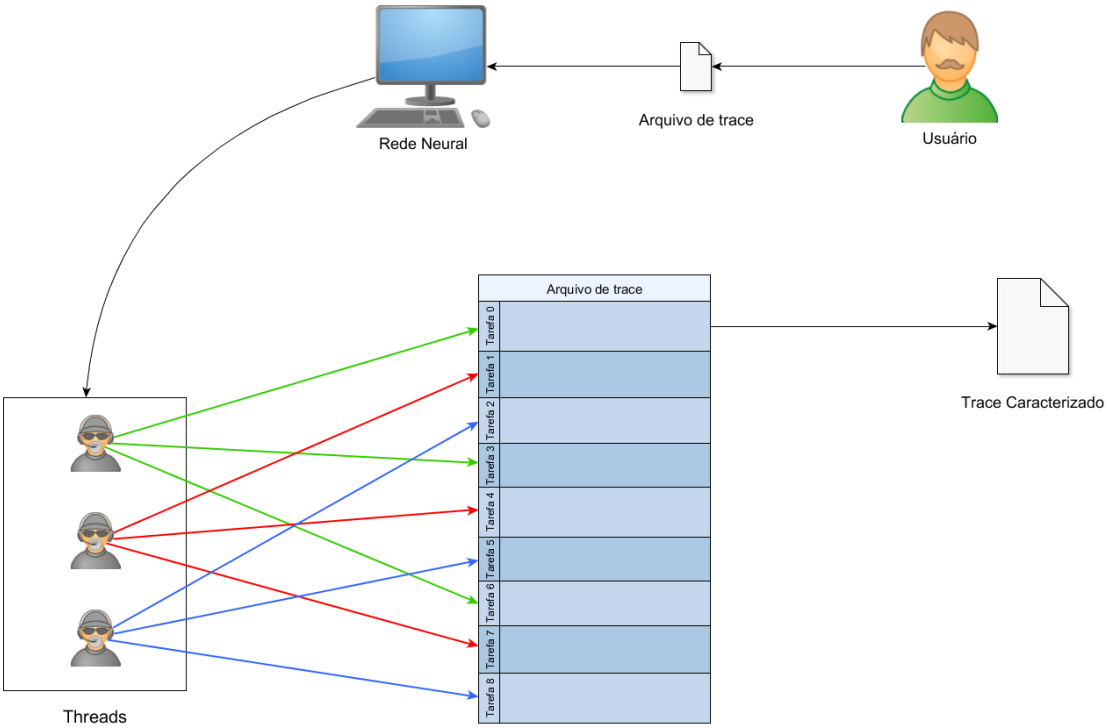

Figura 3.3: Diagrama de execução da RNK Paralela

Capítulo 4

Testes e resultados

Neste capítulo são apresentados os testes realizados a fim de comparar o desempenho das RNKs implementadas nesse projeto e os resultados obtidos.

4.1. Treinamento das Redes

O treinamento das RNKs foi efetuado utilizando um arquivo de *trace* gerado no simulador iSPD (Silva, 2012). Esse arquivo foi criado simulando uma grade de computadores com processadores Intel i3 conectados por uma rede com banda de 100 mbps. Ele é formado por 9 mil tarefas, sendo 3 mil tarefas de computação intensiva, 3 mil de comunicação intensiva e 3 mil neutras.

O treinamento das duas RNKs é realizado sequencialmente, por que as características de cada neurônio são ajustadas em cada iteração, então não haveria vantagem em paralelizar esse processo. Para cada tarefa contida no arquivo de *trace*, é calculada a porcentagem de tempo utilizada para processamento e comunicação, em seguida é calculada a distância da tarefa para cada neurônio e o que estiver "mais próximo" da tarefa de entrada é ajustado de acordo as equações (2) e (3) do capítulo anterior:

$$porc\_comm_{neurônio_i} = porc\_comm_{neurônio_i} + \eta * (porc\_comm_{tarefa_j} - porc\_comm_{neurônio_i})$$

$$porc\_comp_{neurônio_i} = porc\_comp_{neurônio_i} + \eta * (porc\_comp_{tarefa_j} - porc\_comp_{neurônio_i})$$

na qual η é a taxa de aprendizagem, inicializada com valor 0,9 e decrescida em cada iteração de acordo com a fórmula apresentada no capítulo anterior.

Quando a taxa de aprendizagem chega a 0,0001, a fase de treinamento termina, pois a variação às características dos neurônios é ínfima e os seus pesos já convergiram.

4.2. Testes de desempenho

Para realizar o teste de desempenho das RNKs foram utilizados arquivos de *trace* com números de tarefas variados, com o objetivo de analisar quanto tempo é

necessário para classifica-los. Os *traces* foram extraídos do PWA (*Parallel Workloads Archive*) (PWA, 2012), que funciona sob coordenação do Prof. Dr. Dror Feitelson, e então simulados no iSPD e convertidos para o formato WMS (*Workload Modelo of Simulation*) com a extensão ".wmsx", padrão do simulador. Foram usados *traces* de 22 mil tarefas até 13 milhões de tarefas.

Os testes foram realizados em uma máquina com sistema operacional Fedora 20, compilador *gcc* 4.8.3 e o *NVIDIA CUDA Toolkit* versão 6.5. O driver de vídeo utilizado foi o *NVIDIA* 340.62. O *laptop* possui um processador Intel Core i7–740QM @1.73GhHz e placa de vídeo *NVIDIA GeForce 460M*.

Utilizando a RNK com processamento sequencial, foi obtido o gráfico a seguir (Figura 4.1). Nele podemos observar um crescimento do tempo de processamento em função do aumento do número de tarefas a serem caracterizadas. Para obter esses tempos, cada arquivo foi caracterizado pela RNK várias vezes e então foi calculada a média aritmética dos tempos de cada execução.

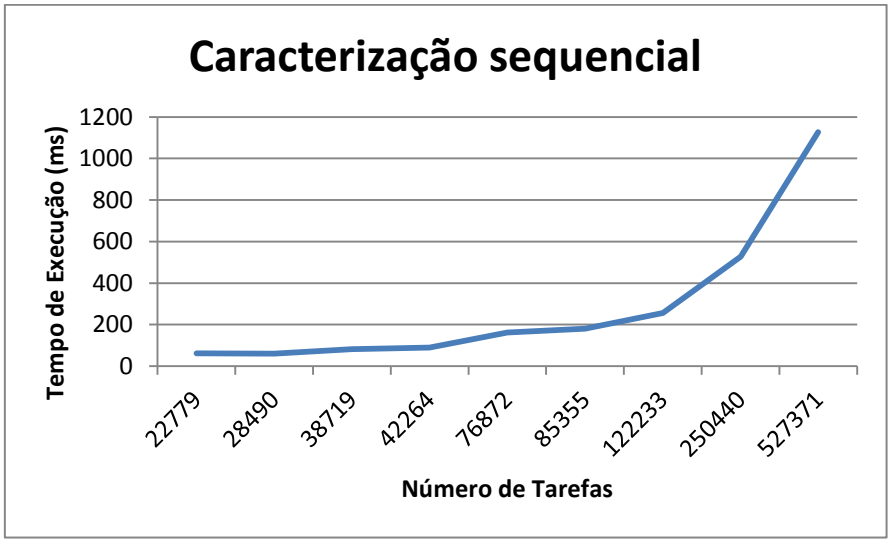

Figura 4.1: Gráfico de tempo do processamento sequencial

Após encontrar o tempo utilizado pela RNK com processamento sequencial, foram executados os testes usando a RNK implementada usando CUDA C. Nessa parte dos testes foram utilizados números diferentes de *threads* para realizar a classificação a fim de explorar um possível aumento de desempenho em função desse número. O gráfico obtido (Figura 4.2) mostrou que há pouca variação no desempenho alterando apenas o número de *threads* utilizado.

Apesar de não haver grande variação entre as execuções variando o número de *threads* utilizados por bloco, foi observado que quanto maior o número de tarefas no arquivo de *trace*, melhor o desempenho da RNK implementada em CUDA C (Figura 4.3). Essa variação fica mais evidente em função do tamanho dos arquivos utilizados. Esse gráfico nos mostra que se forem utilizados *traces* maiores, o ganho de desempenho será ainda maior. A Tabela 4.1 contém os resultados obtidos em cada teste.

	64 threads	128 threads	256 threads	512 threads	1024 threads	Sequencial
22779	57,71	61,19	65,64	57,97	58,67	62,57
28490	58,15	57,62	61,58	58,73	60,94	60,96
38719	82,42	77,51	77,80	77,52	81,12	82,24
42264	83,38	83,94	88,03	83,92	83,02	89,65
76872	151,14	151,61	163,30	159,04	151,25	162,68
85355	171,72	167,53	175,84	167,49	166,85	181,01
122233	239,23	246,24	244,13	239,50	239,07	255,98
250440	499,18	495,37	523,04	490,61	489,59	527,57
527371	1033,42	1037,06	1085,32	1034,69	1031,07	1126,39
1195242	2344,14	2354,52	2421,20	2344,22	2344,07	2547,92
9054066	17795,41	17912,73	18078,25	17860,36	17907,52	19570,45
13313793	26156,36	26288,46	25960,20	26372,51	26196,73	28648,46

Tabela 4.1: Resultados obtidos no teste de desempenho (Número de tarefas x Tempo de Execução)

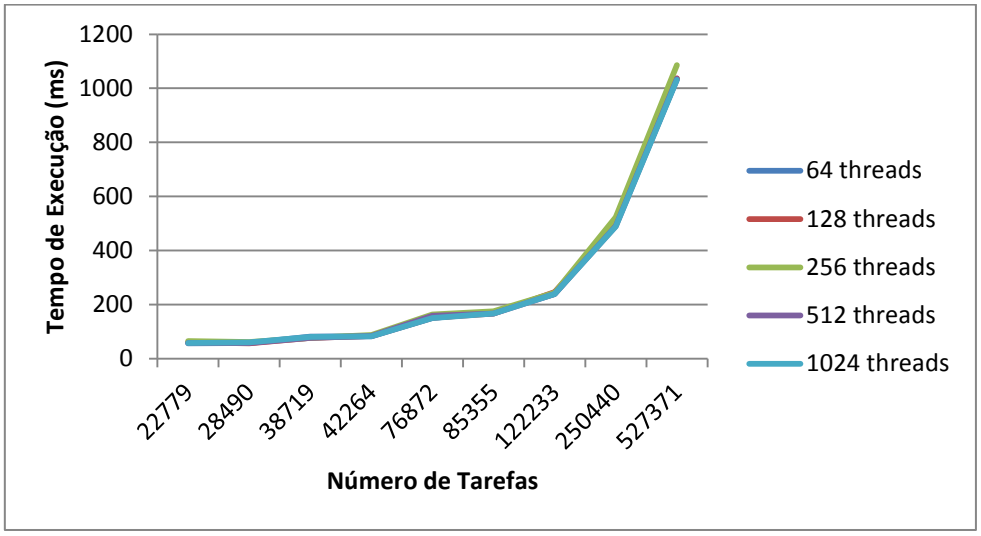

Figura 4.2: Gráfico de tempo de processamento paralelo

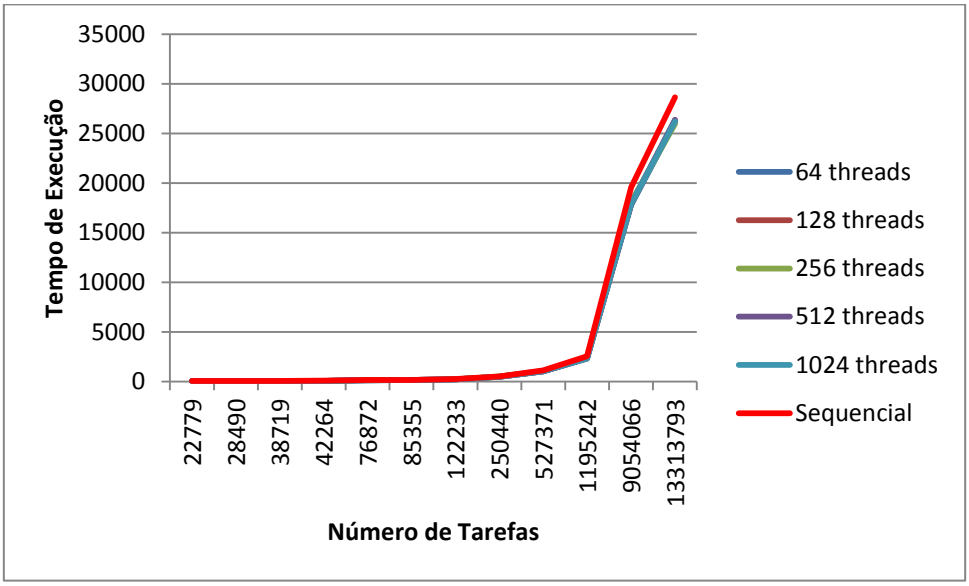

Figura 4.3: Gráfico comparativo

4.2. Testes de precisão

Além do teste de desempenho, foi necessário analisar a precisão das caracterizações realizadas pelas RNKs. Para isso foram criados 6 arquivos de *trace* no iSPD, sendo 2 arquivos para cada característica determinada no capítulo 3. Na simulação foram utilizadas máquinas virtuais com a capacidade de processamento de um Intel i3, com 29750 Mflops/s e uma rede com 100 Mbps de banda. O objetivo desse teste foi conferir se as RNKs foram treinadas corretamente, forneciam resultados úteis e qual a porcentagem de erro. Na tabela 4.1 vemos as características dos arquivos gerados para os testes.

A primeira fase de caracterização foi feita em cada tarefa através da análise do tamanho computacional e do tamanho de comunicação de cada uma, determinando a característica individual de cada tarefa. Depois disso foi calculada a porcentagem de cada tipo de tarefa no arquivo de *trace* e ele foi caracterizado de acordo com o traço mais evidente encontrado. Os resultados obtidos com a caracterização das tarefas são apresentados na tabela 4.2.

	Tarefas Comm. Int.	Tarefas Comp. Int.	Tarefas Neutras
comint0.wmsx	60%	20%	20%
comint00.wmsx	60%	20%	20%
cpuint0.wmsx	20%	60%	20%
cpuint00.wmsx	20%	60%	20%
neutro0.wmsx	20%	20%	60%
neutro00.wmsx	20%	20%	60%

Tabela 4.2: Configuração de arquivos com Comunicação Intensiva

	Tarefas Comm. Int.	Tarefas Comp. Int.	Tarefas Neutras
comint0.wmsx	50%	21%	29%
comint00.wmsx	50%	20%	30%
cpuint0.wmsx	14%	60%	26%
cpuint00.wmsx	18%	60%	22%
neutro0.wmsx	18%	20%	62%
neutro00.wmsx	18%	20%	62%

Tabela 4.3: Caracterização dos *traces*

Podemos notar nessa tabela a caracterização não obteve 100% de acertos e que os erros de caracterização se concentraram em arquivos com maior número de tarefas de Comunicação Intensiva, caracterizando-as como Neutras. Esse erro pode ser atribuído ao grupo de *traces* usado no treinamento da RNK. Na tabela 4.3 são apresentadas as porcentagens de acerto e se a caracterização do arquivo foi feita corretamente.

	Acertos	Erros	Caracterização certa?
comint0.wmsx	90%	10%	Sim
comint00.wmsx	90%	10%	Sim
cpuint0.wmsx	94%	6%	Sim
cpuint00.wmsx	98%	2%	Sim
neutro0.wmsx	98%	2%	Sim
neutro00.wmsx	98%	2%	Sim

Tabela 4.4: Porcentagem de acertos na caracterização

Embora a caracterização das tarefas tenha tido um erro máximo de 10%, a caracterização do arquivo de *trace* obteve 100% de acertos. O erro médio obtido nos testes foi de 5,34%, considerado aceitável para o projeto.

Capítulo 5

Conclusões e Direções

Após a análise dos testes podemos concluir que a paralelização do processamento da Rede Neural de Kohonen utilizando a tecnologia *NVIDIA CUDA* é vantajosa e acelera consideravelmente o desempenho para a caracterização de grandes arquivos de *traces*. Foi possível concluir também que, devido à complexidade do projeto, números variados de *threads* por bloco não influenciam o desempenho da RNK paralela significantemente. Utilizando o grupo de *traces* escolhidos para o treinamento, foi possível obter uma RNK com 5% de erro médio de caracterização, considerado aceitável. Com esses resultados, torna-se possível a utilização de redes neurais para a caracterização de *traces* gerados pelo simulador de grades de computadores iSPD, e portanto aumentar a eficiência da simulação e da utilização desses *traces*.

Com a execução dos testes, pudemos notar que a RNK paralelizada utilizando CUDA se torna mais eficiente que a sequencial para arquivos de *trace* com mais de um milhão de tarefas. Como arquivos com esse tamanho são comuns, torna-se interessante o uso da RNK paralela. Com uma média de erros de 5,34%, as RNKs apresentaram um resultado aceitável ao caracterizar as tarefas e não errou nenhuma caracterização de arquivo de *trace*.

Em relação às dificuldades encontradas no desenvolvimento do projeto, devido às diferenças existentes entre os paradigmas de programação sequencial e paralelo – inclusive o uso do paradigma GP-GPU – uma das maiores dificuldades foi compreender o paradigma paralelo e realizar o mapeamento do algoritmo a fim satisfazer a necessidade apresentada nesse trabalho. Outra situação que apresentou um desafio interessante foi compreender o funcionamento e as possíveis estruturas de uma rede neural.

5.1. Trabalhos Futuros

Como trabalhos sugeridos para o futuro temos a busca por um grupo de *traces* para treinamento da RNK que diminua a porcentagem de erros de caracterização ainda mais, a integração das RNKs implementadas nesse trabalho ao simulador iSPD, para que, ao final de cada simulação, a caracterização do arquivo de *trace* já seja realizada. Também é sugerida a atualização do código à medida que novas ferramentas surjam ou novos parâmetros se tornem relevantes para a caracterização dos *traces*.

Referências Bibliográficas

Bako, L.; Kolcsar, A.; Brassai, S.; Marton, L.; Losonczi, L. (2012) **Neuromorphic Neural Network Parallelization on CUDA Compatible GPU for EEG Signal Classification**. Computer Modeling and Simulation (EMS), 2012 Sixth UKSim/AMSS European Symposium on , vol., no., pp.359,364, 14-16 Nov. 2012

Branco, K. R. L. J. C. (2004). **Índices de Carga e Desempenho em Ambientes Paralelos / Distribuídos - Modelagem e Métricas.** PhD thesis, Universidade de São Paulo (USP), São Carlos.

Calzarossa, M.; Serazzi, G., (1993) **Workload characterization: a survey.** Proceedings of the IEEE , vol.81, no.8, pp.1136,1150, Aug 1993.

Elnaffar, S.; Martin, P. (2006) **Techniques and a Framework for Characterizing Computer Systems' Workloads**. Innovations in Information Technology, 2006 , vol., no., pp.1,5, Nov. 2006

Feitelson, D. G.. (2011) **Workload Modeling for Computer Systems Performance Evaluation.** Disponível em http://www.cs.huji.ac.il/˜feit/wlmod/. [Online; Acessado em: 7-janeiro-2015]

Guthikonda, S. M. (2005). **Kohonen Self-Organizing Maps**. Wittenberg University.

Haykin, S.. (1998). **Neural Networks: A Comprehensive Foundation** (2nd ed.). Prentice Hall PTR, Upper Saddle River, NJ, USA.

Jardim, C. M. G. (2004). **Characterization of Workload Suites for Performance Evaluation**. In 5th Internal Conference on Computer Architecture (ICCA'04).

Manacero, A.; Lobato. R. S.; Oliveira, P. H. M. A.; Garcia, M. A. B. A.; Guerra, A. I.; Aoqui, V.; Menezes, D.; Da Silva, D. T.. 2012**. iSPD: an iconic-based modeling simulator for distributed grids.** In Proceedings of the 45th Annual Simulation Symposium (ANSS '12). Society for Computer Simulation International, San Diego, CA, USA, , Article 5 , 8 pages.

NNWJ, (2004). **Neural Networks With Java**. Disponível em http://www.nnwj.de/kohonen-feature-map.html [Online; acessado em: 7-janeiro-2015]

NVIDIA. **CUDA C Programming Guide.** 2014. Versão: CUDA Toolkit Version 6.5.

NVIDIA. **Parallel Programming and Computing Platform.** 2014. [Online; acessado em: 7-janeiro-2015]. Disponível em: <http://www.nvidia.com/object-/cuda home new.html>.

Pacheco, P.. 2011. **An Introduction to Parallel Programming** (1st ed.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Pendlebury, J.; Huanhuan Xiong; Walshe, R. (2012) **Artificial Neural Network Simulation on CUDA**. Distributed Simulation and Real Time Applications (DS-RT), 2012 IEEE/ACM 16th International Symposium on , vol., no., pp.228,233, 25-27 Oct. 2012

PWA, P. W. A. (2012g). **The standard workload format.** Disponível em http://www.cs.huji.ac.il/labs/parallel/workload/ [Online; acessado em: 7-janeiro-2015].

Sanders, J.; Kandrot, E.. 2010**. CUDA by Example: An Introduction to General-Purpose GPU Programming** (1st ed.). Addison-Wesley Professional.

SEWO, J; SILVA, P. R. R. da. (2003) **Rede Neural Treinada com Algoritmo Não Supervisionado no Contexto de Reconhecimento de Padrões.** – Monografia, Universidade Federal de Mato Grosso do Sul (UFMS), Campo Grande.

Silva, D. T. da. (2012**) Implementação de um Banco de *Traces* de Cargas de Trabalho para o iSPD. –** Monografia, Universidade Estadual Paulista "Júlio de Mesquita Filho" (UNESP), São José do Rio Preto.

Uetz, R.; Behnke, S. (2009) **Large-scale object recognition with CUDA-accelerated hierarchical neural networks.** Intelligent Computing and Intelligent Systems, 2009. ICIS 2009. IEEE International Conference on , vol.1, no., pp.536,541, 20-22 Nov. 2009

Zhao, Y., Shao, G., and Yang, G. (2008). **A survey of methods and applications for trace analysis in grid systems.** In Proceedings of the The Third ChinaGrid Annual Conference (chinagrid 2008), CHINAGRID '08, pages 264–271, Washington, DC, USA. IEEE Computer Society.