Thales Alexandre Zirbel Hubner

Net Topo: uma Abstração da Topologia de Rede para Escalonamento para Aplicações Paralelas e Distribuídas

Florianópolis

2019

Thales Alexandre Zirbel Hubner

Net Topo: uma Abstração da Topologia de Rede para Escalonamento para Aplicações Paralelas e Distribuídas

Monografia submetida ao Programa de Graduação em Ciência da Computação para a obtenção do Grau de Bacharel.

Universidade Federal de Santa Catarina

Departamento de Informática e Estatística

Ciência da Computação

Orientador: Vinicius Marino Calvo Torres de Freitas

Coorientador: Márcio Bastos Castro

Florianópolis

2019

Agradecimentos

A minha família, por ter me dado suporte, carinho, felicidade e educação. Sem eles não seria quem sou nem estaria onde estou.

Ao meu orientador e coorientador, por me guiarem, auxiliarem e incentivarem no processo de pesquisa e escrita, por lidarem com meus problemas mais diversos com calma e paciência e por me acompanhar ao longo deste trabalho.

Ao laboratório *Embedded Computing Lab* (ECL) e seus integrantes, pela infraestrutura, pelo apoio, pela compania, pelas ideias e pelas discussões.

A Universidade Federal de Santa Catarina, por oferecer infraestrutura, por ser um ambientes de pesquisa onde pude atuar, por me oferecer um espaço de ensino e crescimento pessoal, sem o qual não teria realizado este trabalho.

A banca, pelas revisões, pelas críticas e pela contribuição a este trabalho.

Ao corpo docente do INE, pelo incentivo, dedicação e auxilio ao seus alunos, por me muniram do conhecimento necessário na área.

Aos meus amigos e colegas de curso, que fizeram minha experiência universitária possível, me auxiliaram em momentos complicados, me mantiveram feliz, me ajudaram a crescer e me trouxeram bem estar e diversão.

E a muitos outros que também suportaram minha qualidade de espirito e de vida mas não foram citados aqui.

Resumo

A repartição de trabalho em ambientes distribuídos é um problema relevante em aplicações como simulações sísmicas, dinâmica molecular e previsão de tempo, pois estas possuem comportamento dinâmico, gerando desbalanceamento de carga no sistema. Uma das maneiras de resolver esse problema é a utilização de balanceadores de carga dinâmicos, cuja função é reduzir o tempo de execução da aplicação através de uma distribuição de tarefas mais homogênea. Uma alocação de tarefas que leva em consideração a topologia da rede pode reduzir latências de comunicação e efeitos de contenção alocando tarefas que se comunicam próximas uma da outra. Entretanto, poucos balanceadores consideram a topologia do sistema de maneira dinâmica, devido à dificuldade do balanceador de obter informações de topologia de rede, como distância e proximidade de tarefas, e de levá-las em consideração para o remapeamento de tarefas.

Este trabalho desenvolveu a Net Topo, uma abstração da topologia de rede que facilita o acesso a informações de topologia da rede e a utilização destas para distribuição de tarefas. Ela oferece funções de proximidade e distância entre nós na rede com foco em melhorar a repartição de trabalho. Foram implementadas algumas opções de inicialização da estrutura e foi proposta uma forma de arquivar a informação de topologia.

A comprovação do funcionamento da Net Topo se deu através de testes com um *benchmark* sintético para balanceamento de carga. Uma estratégia de balanceamento de carga foi modificada para utilizar a abstração elaborada e, quando comparada com a original, não só teve sobrecusto negligenciável mas apresentou benefícios na inicialização e na velocidade da estrutura. Por arquivar as informações de topologia para execuções futuras, a Net Topo permitiu uma redução de 97% no tempo de inicialização de topologia. O desempenho da abstração foi avaliado em diferentes escalas, mostrando que a Net Topo consegue apresentar boa escalabilidade na maioria das suas funções.

Palavras-chaves: topologia de rede. balanceamento de carga. aplicações distribuídas. aplicações paralelas

Lista de ilustrações

Lista de tabelas

Lista de Algoritmos

Lista de abreviaturas e siglas

PE	*Processing Element*, unidade de processamento
LB	*Load Balancer*, balanceador de carga
hwloc	*Portable Hardware Locality*
netloc	*Portable Network Locality*
CSC	*Compressed Sparse Columns*
XML	*Extensible Markup Language*

Lista de símbolos

S	Aceleração
n	Número de núcleos utilizados
s	Parcela paralelizável do programa
d	Número de dimensões da topologia
k	Número de nós por dimensão da topologia
p	Número de nós totais da topologia
f	Valor de *fanout* da árvore
h	Número de *links* (conexões) remotos por roteador
r	Número de roteadores por grupo
c	Número de cores por máquina
m	Número de máquinas no sistema

Sumário

1 Introdução

Este capítulo apresenta o tema de pesquisa do Trabalho de Conclusão de Curso, o escopo no qual o problema em questão será tratado e a justificativa do projeto. Na Seção 1.1 é apresentado o contexto e a motivação para a realização do trabalho. Na Seção 1.2 são introduzidos os objetivos gerais e específicos, as restrições e premissas existentes, a lista de marcos e os critérios de aceite do projeto. Na Seção 1.3, são abordados os métodos de pesquisa envolvidos para alcançar a solução proposta. Uma visão geral do restante do trabalho é indicada na Seção 1.4.

1.1 Motivação

Aplicações no âmbito científico e industrial incluem cada vez mais detalhes, demandam precisão ainda maior e/ou possuem uma complexidade muito elevada, criando uma demanda cada vez maior por poder de processamento (PILLA, 2014). Uma das maneiras utilizadas para suprir esta demanda é com computação paralela, que soluciona estes problemas através da repartição de trabalho entre unidades de processamento (*Processing Elements* ou PEs) e a execução destas parcelas simultaneamente. Este método permite que um aumento no número de PEs leve a uma redução no tempo da aplicação. Contudo a melhoria no tempo é limitada pela parcela não paralelizável do programa e pelos recursos do sistema (AMDAHL, 1967).

Uma das opções para obter eficiência de processamento é o uso de um número maior de núcleos menos potentes (SNIR, 2011). A organização de um número grande de PEs acaba levando a um sistema com memória distribuída, que operam com o uso de diversas máquinas e utilizam alguma rede para a interligação e comunicação. A escolha destes sistemas é devida ao seu custo baixo e a sua grande escalabilidade através do aumento no número de máquinas. O contraponto destas vantagens é o desenvolvimento destas aplicações, que se torna mais complexo pois tem de levar em consideração fatores impactantes como sincronização, dependência e distribuição de dados, balanceamento de carga e custos de comunicação (PILLA, 2014).

A comunicação em um sistema é realizado através de *links* (ligações) entre PEs. Estas ligações estão dispostas utilizando algum padrão de topologia, como malhas, tori, *fat-trees* ou *dragonfly*, que buscam otimizar alguma característica da rede, como custo de implantação, tráfego ou latência média. A alocação de tarefas nestes sistemas geralmente ocasiona um uso compartilhado de *links* por múltiplos núcleos, que pode saturar partes da rede devido a uma carga maior de informação a ser transmitida do que os *links* suportam, resultando em contenção (BHATELE, 2011). Outro fator de topologia que impacta o desempenho da aplicação é a possibilidade da rede ter ligações com custos heterogêneos (BHATELE et al., 2016), tendo latências diferentes para *links* diferentes.

A repartição de trabalho em ambientes paralelos é um problema a ser tratado, pois aplicações como simulações sísmicas (DUPROS et al., 2010; TESSER et al., 2014), dinâmica molecular (BHATELE; KALÉ; KUMAR, 2009) ou previsão de tempo (RODRIGUES et al., 2010) possuem comportamento dinâmico, ou seja suas cargas mudam ao longo da execução da aplicação, levando um mapeamento homogêneo de tarefas a um estado desbalanceado do sistema, resultando em uma diferença de carga entre alguns PEs. Esta diferença faz com que alguns PEs não tenham trabalho para executar enquanto esperam a conclusão de tarefas nos PEs mais carregados, reduzindo o desempenho geral da aplicação. Para consertar tais desvios pode-se usar um balanceador de carga (*Load Balancer* ou LB) dinâmico, cujo trabalho é remapear as tarefas entre os PEs, buscando um estado mais balanceado do sistema. Este rearranjo garante a utilização dos PEs de maneira mais homogênea e leva a uma redução do tempo da aplicação pela redução de tempo ocioso.

Os caminhos utilizados dentro de uma rede impactam o tempo de execução de um programa, pois caminhos mais lentos resultam em uma latência maior de comunicação e problemas como contenção. É possível que aplicações e LBs levem em conta os custos de comunicação entre os PEs e os custos de movimentação destas tarefas dentro da rede. Tal abordagem pode levar a uma redução do custo de comunicação, evitando a troca de informação e tarefas através de partes lentas da rede. Uma abstração da topologia de rede que permita fácil acesso às informações da rede e de comunicação reduziria a complexidade de criação de aplicações que lidam com contenção e latência.

Hardware Locality (*hwloc*) e *Network Locality* (*netloc*) são duas ferramentas que fornecem uma abstração da topologia para aplicações. Ambas coletam diversas informações de um sistema, realizam uma interpretação intermediária e a disponibilizam para um usuário, de maneira visual ou em um arquivo *Extensible Markup Language* (XML). O *hwloc* apanha somente a parte de topologia de máquina, enquanto o *netloc* coleta informações da rede. Nenhuma das duas visa auxiliar alocação de tarefas com funções para cálculo de proximidade ou distâncias. Este trabalho desenvolveu a Net Topo, uma abstração de topologia com foco em auxiliar na alocação de tarefas, detalhado adiante.

1.2 Objetivos

Os objetivos deste trabalho são de desenvolver e avaliar uma abstração da topologia de rede, permitindo seu uso para balanceadores de carga e aplicações distribuídas. Para este fim, seguem os objetivos específicos:

1. Desenvolver uma API de acesso à informações de topologia de rede;

2. Implementar algoritmos e estruturas para o uso destas informações;

3. Desenvolver uma interface desta abstração utilizando a plataforma Charm++ (University of Illinois, 1996);

4. Modificar um LB para que utilize esta base, visando a prova funcional da abstração.

1.3 Método de Pesquisa

Inicialmente, este trabalho de conclusão foi dividido em duas partes principais: uma de cunho teórico, onde foram estudados os conceitos básicos e o estado da arte para o projeto, e outra mais prática, que envolveu o aprendizado do *framework* Charm++. A primeira parcela foi constituída por um estudo sobre topologia de rede e balanceamento de carga em aplicações paralelas, criando uma base de conhecimentos para o projeto. A segunda parte teve foco no entendimento da plataforma e suas nuances, visando facilitar a implementação da abstração.

A criação da abstração da topologia de rede utiliza a linguagem de programação C++ e é posteriormente inicializada manualmente ou com informações do *framework* de programação paralela Charm++ (University of Illinois, 1996). Esta plataforma possui uma base sólida para a criação, utilização e teste de balanceadores de carga e outras aplicações paralelas (BHATELE, 2010). Além disto, Charm++ contém somente abstrações de topologia fixas (detalhado na seção 2.3.1), carecendo de uma abstração elaborada da topologia de rede, proposta neste trabalho.

Para a criação da abstração de topologia de rede, foi criada uma especificação de suas funções e suas estruturas, criando um esqueleto que foi preenchido posteriormente. Um estudo de estruturas e algoritmos utilizados no estado da arte foi realizado para observar como tais funções podem ser implementadas de maneira eficiente.

Após a implementação dos algoritmos e estruturas da abstração de topologia de rede, foi implementado um método de inicialização da estrutura através do *framework* Charm++, que coleta as informações necessárias para a execução de seu *runtime*. Além da inicialização utilizando o Charm++, foi implementado uma inicialização utilizando XML e uma inicialização manual. Para cobrir o funcionamento das funções da abstração independente da forma de inicialização, foram realizados testes de unidade utilizando *Gtest*.

O balanceador de carga *NeighborLB* foi modificado para utilizar esta abstração de rede, demonstrando seu funcionamento e sua utilidade através das funções implementadas e utilizadas. O balanceador foi comparado com o original, verificando o sobrecusto da abstração criada.

Na última etapa deste trabalho foram realizados testes de desempenho dos algoritmos de inicialização, serialização, proximidade, *hops* e distância, avaliando o seu desempenho relativo a entradas diferentes.

1.4 Organização do Trabalho

O restante do trabalho é dividido em quatro capítulos. O Capítulo 2 apresenta alguns conceitos de importância para o trabalho e duas abstrações de topologia existentes. No Capítulo 3 a abstração Net Topo é apresentada, incluindo detalhes da estrutura, das funcionalidades e da implementação. O Capítulo 4 descreve um balanceador de carga modificado e os testes realizados para comprovar o funcionamento da abstração e avaliar sua escalabilidade. O último capítulo apresenta uma conclusão e possíveis trabalhos futuros. O código implementdo pelo trabalho pode ser encontrado em https://github.com/Thaleszh/Net-Topo.

2 Fundamentação Teórica

O desempenho de um aplicação paralela depende de uma multitude de fatores como *hardware*, técnicas de otimização, topologia, arquitetura utilizada, políticas tomadas e comunicação. Na Seção 2.1 são abordadas questões relacionadas a computação paralela, incluindo algumas de suas características e alguns de seus limites. A Seção 2.2 aborda topologias de sistemas paralelos, suas características, seu impacto na aplicação e algumas abstrações de topologia disponíveis. Na Seção 2.3 é tratado o que é um balanceador de carga, que problemas estes buscam resolver e os benefícios de um balanceador ciente de topologia.

2.1 Computação Paralela

Uma aplicação paralela realiza computação de maneira concorrente, utilizando os recursos de processamento disponíveis simultaneamente para a realização destas tarefas. O ganho de desempenho ao se realizar tarefas simultaneamente é pago com uma complexidade maior de *hardware*, arquiteturas e com a necessidade de sincronização, garantias de exclusão, trocas de mensagem e outras técnicas de programação e compilação (KUCK, 2011).

O fim da lei de Moore (HENNESSY; PATTERSON, 2017) e o limite da barreira de potência tornou a redução dos transistores e o aumento na frequência dos processadores insuficiente, abrindo um espaço para os ganhos de arquiteturas paralelas (TANENBAUM, 2014). Em ambientes de múltiplos núcleos se utiliza um número maior de núcleos menos poderosos, em busca de eficiência através de um paralelismo maior e de um consumo de energia mais baixo (SNIR, 2011). Estes sistemas são facilmente escalados através da inclusão de mais núcleos e extensão da rede, mas agregam uma série de complexidades para o desenvolvimento devido a outros fatores, como sincronização, dependência e distribuição de dados, balanceamento de carga e custos de comunicação entre os núcleos (PILLA, 2014).

A relação entre a aceleração de uma aplicação (**S**) com o número de núcleos utilizados (n) e sua parcela paralelizável (s) é expressa pelo argumento de Amdahl na Equação 2.1 (AMDAHL, 1967). É importante constatar que são desconsiderados detalhes como os custos de acesso a memória, envio de mensagens, geração de *threads* e latência de interconexão. A fórmula é então utilizada como uma estimativa otimista para a aceleração e não como uma referência exata, pois o impacto exato não é simples de calcular e pode ser afetado por diversos fatores externos a execução.

$$\boldsymbol{S} = \frac{1}{(1-s) + \frac{s}{n}} \tag{2.1}$$

Na Equação 2.1, quando n tende ao infinito a equação indica um limite na aceleração de acordo com a parcela serial do programa $(1 - s)$, pois esta não se beneficia da paralelização. Levando este limite em conta, as aplicações paralelas muitas vezes são adaptadas para que sua parte paralelizável inclua mais detalhes ou mais precisão, fazendo com que um aumento no número de PEs leve a mais informação sendo processada e não a uma aceleração (GUSTAFSON, 1988).

2.1.1 Carga e Comunicação

O processamento a ser realizado em uma aplicação paralela é dividido em tarefas e em cargas. A carga representa a quantidade de processamento que uma tarefa precisa realizar em um único PE. Em um sistema paralelo, múltiplas tarefas são executadas simultaneamente, tendo sua carga distribuída entre os PEs do sistema. Um núcleo que possui uma carga maior que um limiar acima da média é considerado sobrecarregado, e um núcleo que tem menos carga que um limiar abaixo da media é considerado subcarregado. Esta diferença nas cargas faz com que alguns PEs não tenham trabalho para executar enquanto esperam a conclusão de tarefas nos PEs mais carregados, sub-utilizando o sistema. Esta situação é chamada de desbalanceamento.

Em aplicações paralelas, a carga de cada tarefa e sua comunicação pode ser diferente das demais tarefas. Essas diferenças podem se manter constantes ou até mesmo variar ao longo da execução da aplicação paralela. Duas características refletem isso: a *regularidade* e a *dinamicidade*. A regularidade de uma aplicação reflete o quanto a carga ou a comunicação de uma tarefa difere de uma outra tarefa. Uma aplicação com carga ou comunicação regular tem cargas ou comunicação aproximadamente iguais. Quando a carga ou comunicação difere entre tarefas, a aplicação tem carga ou comunicação irregular, respectivamente. A dinamicidade de uma apli-

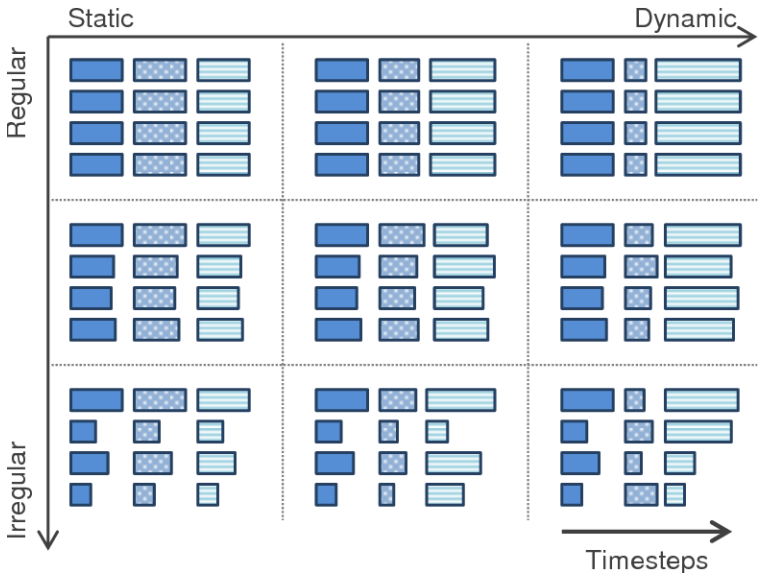


Figura 1 – Diferentes graus de regularidade e dinamicidade em relação a carga de uma aplicação (PILLA, 2014).

cação indica o quanto a comunicação ou carga de uma tarefa muda ao longo do tempo. Quando sua carga ou comunicação não muda ao longo do tempo, é dita estática e quando muda, é dita dinâmica (PILLA, 2014). As Figuras 1 e 2 apresentam a variação de regularidade e dinamicidade em relação a carga e a comunicação de uma tarefa, respectivamente.

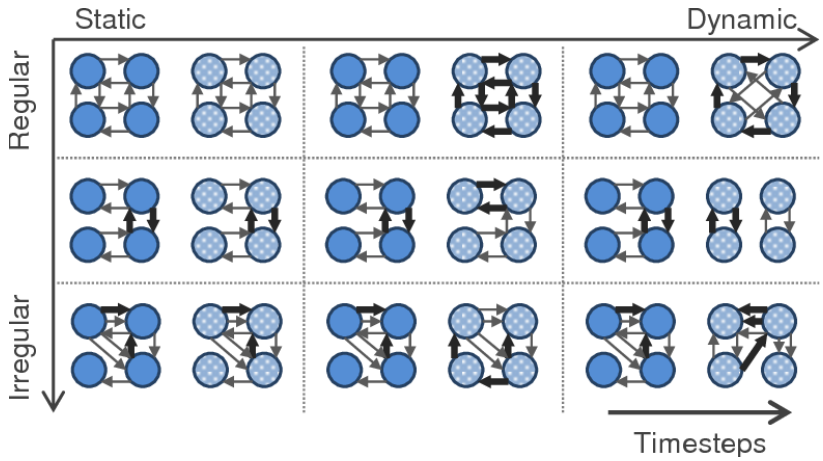


Figura 2 – Diferentes graus de regularidade e dinamicidade em relação a comunicação de uma aplicação (PILLA, 2014).

A distribuição de tarefas em um ambiente paralelo deve levar em consideração sua regularidade e dinamicidade. Técnicas de escalonamento, alocação de tarefas e balanceamento de carga são utilizadas para gerenciar e dispor as tarefas de maneira eficiente no sistema. Na Seção 2.3 serão abordadas algumas técnicas de balanceamento de carga.

2.1.2 Computação Distribuída

Em ambientes multiprocessados, existe algum dispositivo de memória compartilhada entre todos os PEs, o qual é organizado de forma a impedir problemas de coerência e consistência. *Chips multicore* e *manycore* são exemplos de ambientes multiprocessados. Ambientes multiprocessados não escalam bem devido à complexidade de arquitetura e custos de produção, abrindo espaço para ambientes multicomputados. Em ambientes multicomputados a memória não é compartilhada entre todos os PEs e algum meio de interconexão é utilizado para realizar as trocas de informação, mas o sistema ainda é organizado de maneira similar (TANENBAUM, 2014).

Em um sistema distribuído, computadores com diferentes arquiteturas, organizações, locais e comportamentos podem ser interconectados em uma mesma rede e utilizados em conjunto. Devido à alta independência dos componentes, estes sistemas são altamente escaláveis através da adição de computadores e expansão da rede. A comunicação e coordenação destes sistemas se torna muito mais complexa e custosa devido ao número de componentes, precisando ser tratada com mais detalhe (TANENBAUM, 2014).

2.2 Topologia de Rede

Sistemas paralelos utilizam redes de interconexão para efetuar a comunicação entre seus PEs. Estas redes têm propriedades como diâmetro, distância média, grau, latências e largura de banda que regem a comunicação dentro da rede. O padrão de interconexões entre PEs é chamado de topologia e esta pode assumir diversas formas como malhas, tori e *fat-trees*.

As topologias de uma rede podem ser dividas em duas categorias: *diretas* e *indiretas*. Em redes diretas, os PEs são conectados diretamente uns aos outros através de *links*. Para que uma mensagem possa ser transmitida de um PE a outro, esta precisa trafegar através de vários destes *links*. O cruzamento de um *link* é chamado de *hop* e a distância entre um ponto a outro na rede pode ser definido pelo número de *hops* realizados. Malhas, tori e hipercubos são exemplos de redes diretas. Em redes indiretas, PEs são conectados a *switches* (roteadores) que roteiam as mensagens. Nenhum PE é ligado diretamente a outro PE e portanto é necessário cruzar dois ou mais *switches* para alcançar outro PE. *Fat-trees* e *dragonfly* são exemplos de redes indiretas (BHATELE, 2011).

A hierarquia é um fator presente em algumas topologias indiretas, como *fat-tree* e *dragonfly*, e divide esta em níveis de proximidade. Cada nível de hierarquia na topologia representa um agrupamento diferente da rede. Redes hierárquicas apresentam pontos de gargalo de tráfego entre seus níveis e requerem uma replicação de *links* para que isto não se torne um problema.

Uma rede de interconexão tem duas propriedades importantes que afetam sua comunicação: *simetria* e *uniformidade*. Um nível de topologia é dito simétrico quando o tempo de comunicação de um ponto A para um ponto B é o mesmo que de B para A. Caso contrário, este é dito assimétrico. Um nível de topologia é uniforme quando todos os PEs neste nível tem o mesmo tempo de comunicação entre si (PILLA, 2014). Nesse sentido, uma rede indireta pode se tornar assimétrica devido ao roteamento de informações.

Algumas métricas são utilizadas para observar o comportamento de uma rede, como o diâmetro, o número de *links*, o grau e a bisseção de *links*. O diâmetro de uma rede é definido como sendo o maior caminho dentre os caminhos mais curtos que ligam quaisquer dois nós da topologia. Usualmente, as distâncias são medidas através do número de *hops*, sendo esta

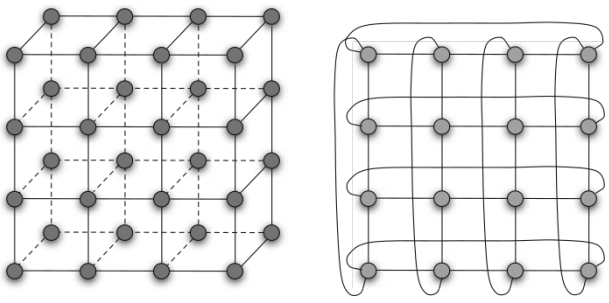


Figura 3 – Uma malha de dimensões (4,4,2) a esquerda e uma torus 2D a direita (BHATELE, 2011)

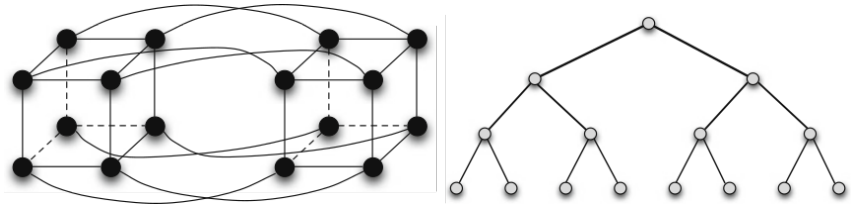


Figura 4 – Uma rede direta em forma de hipercubo de 4 dimensões a direita e uma indireta em forma de *fat-tree* binária com 4 níveis a esquerda (BHATELE, 2011)

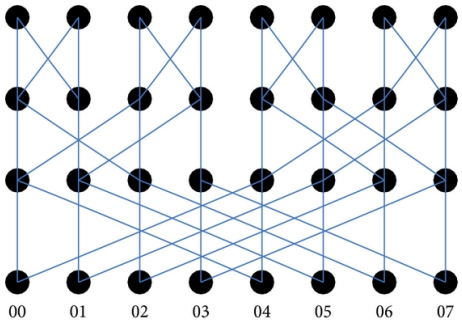


Figura 5 – Uma rede *butterfly* de 3 níveis e 8 nós (LIU et al., 2014)

métrica também utilizada para determinar a maior latência dentro de uma rede. O número de *links* apresenta a quantidade total de *links* dentro da rede, uma informação usada para avaliar o custo de implantação. O grau de uma rede é o número de *links* conectados em cada nó da rede. A bisseção da largura de banda representa a quantidade de *links* entre duas partes da rede, partida de modo a gerar o mínimo de *links* entre duas metades da rede. Essa métrica é utilizada para estimar o quanto de tráfego a rede suporta (CULLER; SINGH; GUPTA, 1998). A seguir, são apresentadas algumas topologias abordadas neste trabalho.

- **Malha**: Uma topologia direta similar a uma matriz de múltiplas dimensões, onde cada *link* só cruza uma dimensão da rede e na menor distância possível (SOLIHIN, 2009). Malhas podem ser dispostas em diversas dimensões. A Figura 3 representa uma malha de dimensões 4, 4 e 2.

- **Torus**: Uma malha com um *link* entre nodos finais de cada dimensão da rede, reduzindo o diâmetro da rede e aumentando sua bisseção. Uma torus de dimensões 4 e 4 pode ser observada na Figura 3.

- **Hipercubo**: Uma malha com 2 nodos em cada dimensão. Um hipercubo visa reduzir a distancia máxima da rede. A Figura 4 mostra um exemplo de hipercubo.

- ***Fat-tree***: Uma topologia de rede indireta em forma de árvore, onde *links* em níveis mais altos possuem mais largura de banda. Esta topologia é encontrada como topologia interna de máquinas. Uma *fat-tree* é representada na Figura 4.

Símbolo	Descrição
d	Número de dimensões da topologia.
k	Número de nós por dimensão da topologia.
p	Número de nós totais da topologia.
f	Valor de *fanout* da árvore.
h	Número de *links* remotos por roteador
r	Número de roteadores por grupo

Tabela 1 – Símbolos utilizados na Tabela 2.

Topologia	Diâmetro	Bisseção	*Links*	Grau
Malha 2-D	$2\sqrt{p}-1$	$\sqrt{p}$	$2\sqrt{p}\times(\sqrt{p}-1)$	$2d$
Malha K-D	$dk-1$	k^{d-1}	$dk^{d-1}\times(k-1)$	$2d$
Tori	$\frac{dk}{2}-1$	k^d	dk^d	$2d$
Hipercubo	$\log_2 p$	$\frac{p}{2}$	$log_2 p\times\frac{p}{2}$	$\log_2 p$
Fat-tree	$2\times\log_f p$	$\frac{p}{2}$	$f(p-1)$	$f+1$
Butterfly	$\log_2 p$	$\frac{p}{2}$	$2p\times\log_2 p$	4
Dragonfly	3	$h((r+2)^2/4)$	$r\times(r-1)+rh$	$r+h$
Slim Fly	2	-	$2\times r\times(r-1)$	$r+h$

Tabela 2 – Diâmetro, bisseção de *links*, número de *links* e grau das topologias Malha, Tori, Hipercubo, *Fat-tree*, *Butterfly*, *Dragonfly* (LI et al., 2016) e *Slim Fly*. Os símbolos utilizados são apresentados na Tabela 1. Adaptado de (SOLIHIN, 2009). A topologia *Slim Fly* não apresenta uma fórmula de bisseção, pois ela é calculada aproximadamente utilizando um particionador. Sua bisseção é maior que uma *dragonfly* mas menor que de um hipercubo (BESTA; HOEFLER, 2014).

- ***Butterfly***: Uma topologia indireta que visa escalabilidade e redução de distâncias através da replicabilidade de *links* e de roteadores. É utilizada para gerar uma bisseção de banda elevada dentro de uma rede. Sua desvantagem é o grande número de *links*, o qual é muito superior ao de outras topologias (CULLER; SINGH; GUPTA, 1998).

- ***Dragonfly***: Uma topologia indireta que cria agrupamentos de roteadores em grupos distintos, visando reduzir o número de *links* da rede para reduzir seu custo.

- ***Slim Fly***: Uma topologia indireta similar a *dragonfly* mas otimizada matematicamente, reduzindo o custo e energia gasta enquanto aumenta a resiliência da rede em relação à falha de *links* (BESTA; HOEFLER, 2014).

A Tabela 2 apresenta algumas topologias e suas características principais: diâmetro, bisseção, *links* e grau. Os símbolos utilizados são descritos na Tabela 1. É importante ressaltar que $p = dk$ nas topologias de Malha ou Tori. Na Tabela 3 são comparadas topologias com 2^{16} nós. A topologia *dragonfly* se destaca, pois tem *links* com latências diferentes e custos de

Topologia	Diâmetro	Bisseção	*Links*	Grau
Malha 4D	63	4096	245760	8
Tori 4D	31	65536	262143	8
Hipercubo	16	32768	524288	16
Fat-tree	16	32768	262143	5
Butterfly	16	32768	2097152	4
Dragonfly	3	4225	16896	129
Slim Fly	3	4225	16896	129

Tabela 3 – Diâmetro, bisseção de banda, número de *links* e grau das topologias de Malha 4D, Tori 4D, Hipercubo, *Fat-tree* de *fanout* 4, *Butterfly* e *Dragonfly*, todas com 2^{16} (65536) nodos.

implementação diferentes, fazendo com que métricas baseadas em *hops* e número de *links* não reflitam a distância e o desempenho da rede.

Conforme cresce o número de PEs de um sistema, o fluxo de informação na rede aumenta e mais recursos de rede (*links* e *switches*) são compartilhados e disputados pelos PEs. Este compartilhamento pode levar a um problema chamado de contenção, onde a latência da rede aumenta devido a saturação dos *links*. Um mapeamento de tarefas ciente de topologia de rede pode reduzir o compartilhamento de recursos de rede e portanto melhorar o desempenho da aplicação (BHATELE, 2011).

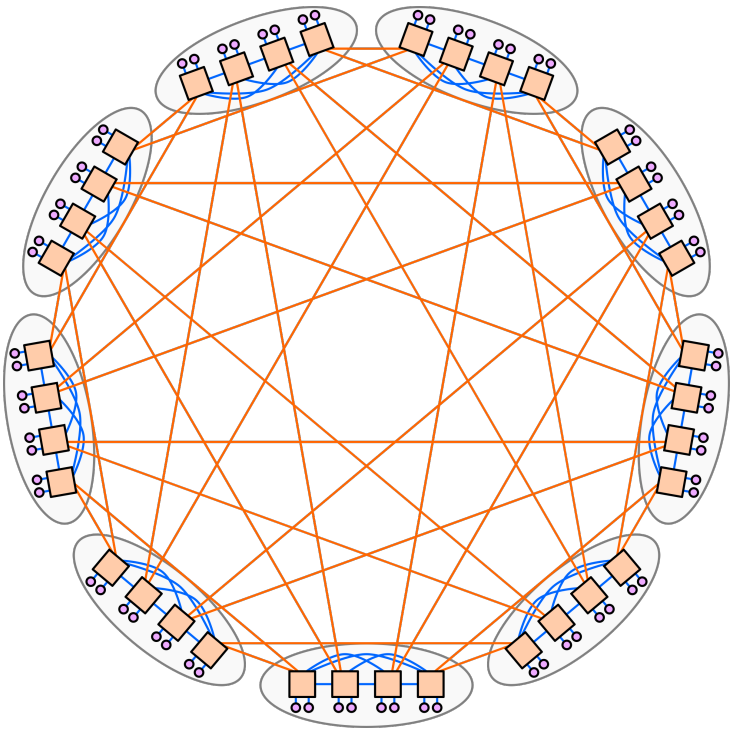


Figura 6 – Um exemplo de rede *dragonfly*. Roteadores e *links* globais em laranja, nós em vermelho e *links* locais em azul. Fonte: Openclipart, <https://openclipart.org/detail/291507/dragonfly-network-topology>, acessado em 08 nov. 2018

Topologias de rede podem ter agrupamentos de máquinas interligados com *links* de maior latência ou menor capacidade, como, por exemplo, a *dragonfly* da Figura 6. A contenção e a alocação de tarefas neste tipo de rede afeta severamente seu desempenho. Nestas redes, uma política adequada de roteamento, de alocação de tarefas, de migração de tarefas e de carga de trabalho paralela é crucial (BHATELE et al., 2016). Roteamento dinâmico e adaptativo é um exemplo de política geralmente acoplado a redes deste tipo, para que decisões de roteamento levem em consideração a atual contenção da rede. Com isso, caminhos menos sobrecarregados são tomados, distribuindo melhor a carga da rede e evitando problemas graves de contenção (KIM et al., 2008). Balanceamento de carga ciente de topologia se torna um fator ainda mais relevante para este tipo de rede.

2.2.1 Abstrações de Topologia

Nesta seção são abordados dois trabalhos que abstraem informações de topologia para seu uso em aplicações: O *Portable Hardware Locality* (*hwloc*) (BROQUEDIS et al., 2010) e o *Portable Network Locality* (*netloc*) (GOGLIN; HURSEY; SQUYRES, 2014).

Hwloc é um *software* que provê uma abstração da hierarquia das arquiteturas de máquinas, dispondo uma série de informações de *cache*, núcleos, *multithreading*, *sockets* e dispositivos de entrada e saída (BROQUEDIS et al., 2010). Seu objetivo é facilitar o acesso de informações complexas das arquiteturas atuais, de maneira uniforme e portável para aplicações. Em sistemas Unix, por exemplo, as informações coletadas por *hwloc* estão em diversos arquivos espalhados pelo sistema. Portanto, diversas leituras de dados em disco são necessárias para adquirir estas informações. Em questão de topologia, o *hwloc* trabalha somente em um ambiente multiprocessado, em uma única máquina, não oferecendo suporte em relação a topologia de um ambiente multicomputado ou distribuído.

Netloc é um projeto de *software* acoplado ao *hwloc* que provê uma abstração da topologia de rede. Assim como o *hwloc*, o *software* é usado para encontrar informações completas da topologia, ainda não descobertas, e disponibilizá-las para o usuário. O trabalho do *netloc* é voltado para métodos de descobrimento e representação da topologia de rede, unindo estas informações com as informações de máquina do *hwloc* (GOGLIN; HURSEY; SQUYRES, 2014).

O projeto *netloc* ainda está em desenvolvimento e ainda não possui representação de distância ou latência entre áreas distintas da rede. Além disso, sua preocupação é de fornecer todas as informações que conseguir sobre a topologia utilizada de maneira dinâmica para considerar alterações na rede devido a falhas e expansões (GOGLIN; HURSEY; SQUYRES, 2014). A quantidade e complexidade destas informações aumentam a ocupação de memória e o tempo de inicialização da abstração, adicionando um custo indesejado.

2.3 Balanceamento de Carga

Balanceamento de carga é uma técnica que realiza distribuição de cargas de computação e comunicação em um sistema de maneira que nenhum processador seja sobrecarregado e que reduza custos de comunicação (BECKER; ZHENG; KALÉ, 2011). Tendo em vista uma aplicação dinâmica, a distribuição de carga e comunicação se altera ao longo da aplicação, sendo necessário realizar um balanceamento de carga periodicamente para mitigar o desbalanceamento.

Encontrar um mapeamento ótimo para uma aplicação paralela executada em múltiplas máquinas idênticas é um problema NP-*hard* (LEUNG; KELLY; ANDERSON, 2004). Como o objetivo de balancear é aumentar o desempenho, utilizam-se heurísticas para realizar um balanceamento sub-ótimo, já que um mapeamento ótimo iria demorar a ponto de degradar o tempo da aplicação (PILLA, 2014).

Um balanceador de carga (*Load Balancer* ou LB) pode administrar informações sobre a distribuição de carga e comunicação de três maneiras: *centralizada*, *distribuída* e *hierárquica*. Um LB centralizado administra a informação do sistema como um todo, podendo realizar decisões mais completas, mas criando um gargalo de informações. Um LB distribuído é o outro extremo pois observa somente informações locais, tendo um processo de balanceamento mais escalável, mais rápido porém menos efetivo. Uma abordagem hierárquica mistura as duas outras abordagens em camadas hierárquicas diferentes (BECKER; ZHENG; KALÉ, 2011).

Um balanceamento de carga pode utilizar informações da topologia da rede para posicionar tarefas que se comunicam próximas umas das outras, reduzindo a latência de comunicação e efeitos de contenção na rede. Esta informação se torna ainda mais relevante quando o diâmetro de uma rede é grande, quando esta possui assimetria ou latências diferentes para partes diferentes da rede (BHATELE et al., 2016). Um LB que utiliza estas informações é dito ciente de topologia.

2.3.1 Charm++

Charm++ (University of Illinois, 1996) é um sistema de programação paralela independente de máquina desenvolvido na *University of Illinois at Urbana-Champaign* (KALE; BHATELE, 2013). O sistema é baseado na ideia de objetos migráveis chamado de *chares* cuja execução é guiada pela troca de mensagens. O seu sistema de execução (*runtime*) possibilita a execução de programas paralelos com mecanismos avançados, como balanceamento de carga automático e *checkpointing*. Charm++ suporta tanto ambientes de multiprocessadores quanto de multicomputadores, conseguindo realizar comunicação através de memória compartilhada e de protocolos ou de infraestruturas de comunicação em rede, tais como UDP, MPI, OFI, Infiniband, uGNI e PAMI (PILLA; MENESES, 2015).

Chares são objetos concorrentes utilizados no *framework* Charm++ que possuem so-

mente métodos de entrada visíveis, que são utilizados de maneira remota e assíncrona (KALE; BHATELE, 2013). Com estes métodos, *chares* podem se comunicar remotamente sem necessidade de barreiras. Pela sua natureza desacoplável, *chares* são altamente migráveis, podendo trocar de PEs com facilidade.

Para garantir a eficiência na troca de mensagens entre suas tarefas, a plataforma Charm++ adquire as informações de topologia da rede com o seu *runtime*. No início da execução de uma aplicação, é realizado uma troca de mensagens entre os processos em execução em busca de suas conexões. Esta informação é utilizada para inferir a topologia utilizada dentre a seguinte lista: *mesh*, *fat-tree*, torus, anel e grafo completo.

2.4 Conclusão

Em ambientes paralelos e distribuídos, informações da topologia podem ser utilizadas para melhorar o desempenho das aplicações. Alocação de tarefas é uma das técnicas que se beneficia disto, podendo reduzir a latência de comunicação e a contenção de uma rede através de alocação próxima de tarefas que se comuniquem. Estes fatores se tornam mais impactantes quando se considera redes indiretas e com custos de comunicação diferenciados entre partes da rede.

A plataforma Charm++ provê um ambiente para programação paralela com a possibilidade de utilização de balanceadores de carga. Porém, as informações topológicas adquiridas pela plataforma são imprecisas e apresentam limitações. O próximo capítulo aborda a abstração proposta por este trabalho, Net Topo, que visa facilitar o acesso de algumas informações de topologia.

3 Net Topo

A contenção de rede e a latência de comunicação são dois fatores ligados à topologia de rede que afetam diretamente o desempenho de uma aplicação paralela e distribuída. Estes dois fatores podem ser reduzidos com uma distribuição de carga que aloque tarefas que se comuniquem próximas uma da outra (BHATELE, 2011). Para saber o quão próximo um PE está de outro e realizar uma alocação eficiente, é preciso obter informações sobre a topologia da rede sendo utilizada. Ferramentas para descobrimento e uso de informações de topologia existem, mas têm suas limitações.

A ferramenta *hwloc* realiza o descobrimento e armazenamento de topologias de máquina, oferecendo um detalhamento profundo de qualquer informação da máquina, mas sem ter qualquer informação inter-máquina. O projeto *netloc* pretende estender o *hwloc* adquirindo informações completas da camada de rede, porém está em desenvolvimento e, justamente por procurar um conjunto de informações grande, se torna caro em inicialização e em espaço de memória. O *runtime* Charm++ possui uma série de maneiras para utilizar uma topologia de rede, mas estas são enquadradas em alguns tipos de topologias pré-definidas, gerando algumas imprecisões.

Neste capítulo é abordado a Net Topo, uma abstração da topologia de rede criada neste trabalho que visa auxiliar escalonadores e balanceadores de carga cientes de topologia. Os objetivos da abstração são abordados na Seção 3.1. A Seção 3.2 apresenta as estruturas cogitadas e utilizadas na abstração. Por último, as funcionalidades da abstração são descritas na Seção 3.3.

3.1 Objetivos da Net Topo

O principal objetivo da Net Topo é auxiliar na alocação e distribuição de carga fornecendo informações de uma topologia de rede. Para que a abstração possa ser utilizada independente da topologia de um sistema, sua representação deve ser flexível e genérica, podendo comportar tipos diferentes de topologias sem perder funcionalidades. Com isso, a abstração pode ser utilizada em escalonadores e balanceadores independentemente do sistema em que estejam.

O principal uso das informações de topologia para um escalonador ou balanceador de carga é indicar a proximidade entre tarefas, visando a redução de latência e contenção. Para suprir esta informação, a abstração deve poder indicar a proximidade entre dois PEs na rede e suprir informações de distância como *hops* ou latência.

Para acelerar o desenvolvimento e auxiliar escalonadores e balanceadores de carga, é importante que a abstração ofereça métodos úteis e não gere um sobrecusto que degrade o

desempenho da aplicação. Em outras palavras, a Net Topo não pode retardar demasiadamente o processo de distribuição de carga.

Um outro objetivo da abstração é oferecer persistência de informação entre execuções no mesmo sistema, seguindo a ideia do *hwloc* de arquivar uma topologia. Esse arquivamento permite uma importação posterior das informações de topologia sem passar novamente por um processo de descobrimento, reduzindo o sobrecusto de inicialização da estrutura (BROQUEDIS et al., 2010). Outro benefício é que o arquivo pode ser repassado para outras máquinas que tenham a mesma topologia de rede, para que não tenham que realizar o descobrimento da topologia.

A plataforma Charm++ oferece suporte para balanceadores de carga e oferece uma estrutura que facilita o desenvolvimento de um balanceador de carga. Um último objetivo da Net Topo é a integração com o módulo de balanceamento de carga do Charm++. Acoplar a abstração junto ao Charm++ aumenta a aplicabilidade da abstração e melhora o uso de uma plataforma já estabelecida.

3.2 Estrutura

Um grafo é uma maneira de representar informação como um conjunto de vértices e arestas que associam dois vértices. Uma topologia de rede simétrica pode ser representada por um grafo não direcionado $G = (V, E)$ onde os vértices V são PEs e as arestas E são os *links* entre PEs (SOLIHIN, 2009). Para representar custo ou latência de um *link*, é adicionado um peso w a cada aresta E. A forma de armazenamento das informações deste grafo influencia diretamente a forma de acesso aos dados e a quantidade de memória alocada no sistema.

Portanto, foram estudadas três abordagens de se armazenar as informações do grafo de topologia: (i) um armazenamento de todas as suas distâncias; (ii) um armazenamento do grafo de topologia; ou (iii) ambos.

O armazenamento de distâncias visa capturar somente a distância entre PEs, obtendo um acesso rápido e simplificado. Para obter estas distâncias, necessita de um tempo maior de inicialização. Esta opção gera menor ocupação de memória, pois armazena somente uma parcela sintetizada das informações. Por outro lado, como não observa o agrupamento e a disposição da topologia, impedindo a utilização destas para distribuição de carga e em cálculos de proximidade.

A opção de armazenar o grafo visa manter as informações relevantes para distribuição de carga, mantendo a possibilidade de obter distâncias. Quando comparada com a opção de armazenar somente distância, ocupa mais memória e é mais lenta no acesso de distâncias, precisando iterar sobre a estrutura para as obter. Para acelerar a obtenção destas distâncias é possível utilizar mecanismos de programação dinâmica, como uma *cache*, para manter uma parcela das

distâncias calculadas.

A opção de guardar tanto as distâncias calculadas quando a estrutura geral de topologia gera uma ocupação de memória agregada mas obtém os benefícios de ambas. Esta opção foi a escolhida para este trabalho. Ela permite utilizar as informações da topologia, realizar um cálculo de distâncias rápido e não impede extensões futuras. A penalidade desta opção é uma utilização maior de memória. O armazenamento completo das distâncias em vez de uma *cache* é devido a persistência de informações entre execuções, onde a *cache* não necessariamente manteria localidade devido a padrões diferentes de comunicação e distribuição.

Existem diversas estruturas de dados para armazenar as informações de um grafo, e seus benefícios são dependentes de como esta estrutura é utilizada. O grafo de topologia sendo armazenado é dado por $G = (V, E, W)$, onde $V = \{$ união dos $v \in V$, que representam PEs$\}$, $E = \{$ união das relações $(v, u) \mid v, u \in V\}$, W é uma função de peso $W : v, u \rightarrow \mathbb{R}$. Foram avaliadas quatro estruturas de dados para o armazenamento de G, observando suas vantagens, desvantagens e aplicabilidade para este projeto. Elas são descritas a seguir.

- **Lista de listas**: Uma representação que dispõe V em uma lista L e para cada $v \in V$ há uma lista de arestas $L_v \mid \forall e \in E, e \in L_v \iff v \in e$. É uma estrutura simples que tem tempo de acesso de $O(\mid L_v \mid)$. Ocupa memória em complexidade $O(\mid V \mid \times \mid L_v \mid)$. Esta estrutura é utilizada no *framework* de balanceamento de carga do Charm++.

- **Árvore**: Uma árvore representando uma hierarquia de proximidade. Tem tempo de acesso de $O(log_f \mid V \mid)$ e ocupa memória relativa a $O(\mid V \mid)$. A base do logaritmo f depende do número de divisões por camada da árvore (chamado de *fanout*). É utilizada no *hwloc* para a estrutura topológica interna de uma máquina. No *hwloc* os PEs se encontram como folhas da árvore e são agrupados de acordo com a proximidade dentro da máquina. Esta abordagem se adéqua bem para topologia interna devido a sua hierarquia inerente, mas não descreve bem uma topologia não hierárquica, como uma torus, pois muitas máquinas se encontram no mesmo nível hierárquico, possuem arestas entre si e não necessariamente contém uma hierarquia com mesmo *fanout* ou profundidade. Uma árvore não tem inserção de ligação.

- **Matriz**: Uma matriz representando todas as associações possíveis entre os PEs da rede. Tem tempo constante de acesso a uma ligação mas ocupa memória quadrática em relação a $\mid V \mid$. Uma das vantagens de uma matriz é sua alta mutabilidade de ligações, sendo possível alteração ou inserção posteriores de ligações na topologia com tempo constante.

- ***Compressed Sparse Columns*** (CSC): uma representação construída a partir de uma matriz, retirando todas as ligações inexistentes para economia de memória. A estrutura é organizada em três listas: uma lista com os valores não nulos da matriz, uma lista com as linhas destes valores e uma lista que indexa o início das linhas de cada coluna. Utiliza

Estrutura	Leitura	Memória utilizada	Inserção de ligação
Listas de Listas	$O(g)$	$O(p \times g)$	$O(2g)$
Matriz	$O(1)$	$O(p^2)$	$O(1)$
Árvore	$O(log_f p)$	$O(p)$	$*$
CSC	$O(g)$	$O(p + l)$	$O(p + l)$

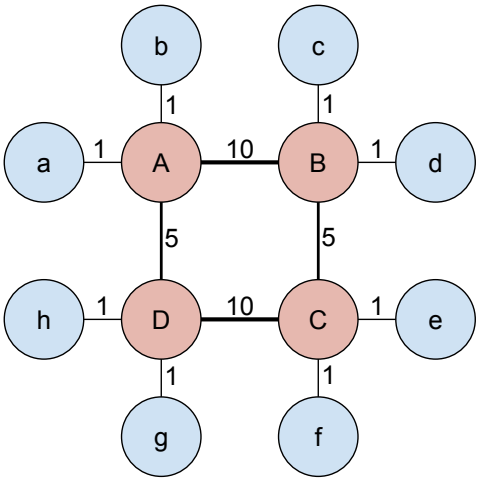
Tabela 4 – Complexidade de leitura e de memória das estruturas cogitadas em relação ao número de vértices $p =\mid V \mid$, ao número de arestas $l =\mid E \mid$ e ao grau g da topologia. A estrutura de Árvore depende também de um fator de *fanout* f.

índices de acesso similar a uma lista de listas. É feita de maneira a otimizar localidade temporal de memoria quando é realizado uma travessia do grafo. Um CSC também pode ser distribuído de maneira reduzida, sem prejudicar um algoritmo de travessia de grafos (SUN; VANDIERENDONCK; NIKOLOPOULOS, 2017). O problema de uma CSC é sua baixa mutabilidade, necessitando uma refatoração para inserir novas arestas ou de um número de espaços vazios na sua inicialização, que podem não ser o suficiente.

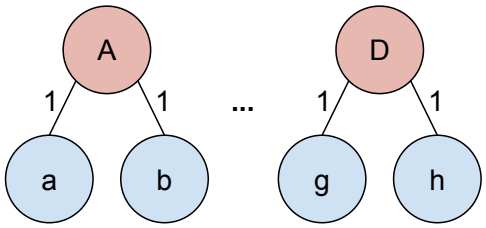
Na Tabela 4 é observada a complexidade do tempo de leitura de uma ligação específica da estrutura, de tamanho ocupado na memória e de inserção de uma ligação, todos em relação ao número de vértices $p =\mid V \mid$, ao número de arestas $l =\mid E \mid$ e ao grau g da topologia.

Uma estrutura puramente topológica não proveria um acesso rápido para distância, pois teria de calculá-la a cada chamada. Para acelerar o acesso às distâncias, pode-se usar uma técnica de programação dinâmica denominada *memoização*. A técnica consiste em armazenar uma informação indireta em uma estrutura, para que não tenha que ser calculada novamente. Uma estrutura somente de distâncias, se usada como um mecanismo de memoização, iria ser ampliada até que em algum momento conteria todas as distâncias possíveis, onde $g = p$ e $p = l^2$. Quando isto ocorrer, todas as estruturas apresentadas ocupariam memória quadrática em relação a p, exceto uma árvore, que não oferece um acesso constante à distância. Dentre as estruturas de dados apresentadas, a matriz é a com acesso mais rápido neste caso, sendo a estrutura candidata para memoização completa. Esta abordagem não escalaria com o número de PEs. Realizar uma abordagem com ambas as estruturas resultaria num uso extenso da memória, algo indesejável para uma aplicação distribuída e escalável.

Para alcançar escalabilidade, foi utilizada uma estrutura inspirada no *netloc*, dividindo a topologia em dois níveis: uma camada hierárquica em árvore que representa a topologia interna das máquinas, similar ao *hwloc*, e outra camada que apresenta as conexões em um nível de rede, representada com CSC pela sua utilidade em travessia de grafos. Uma matriz de distâncias foi utilizada como mecanismo de memoização para distâncias no nível de rede, reduzindo seu tamanho por considerar somente um nível de hierarquia. Além disto, foi assumido simetria no nível de rede, possibilitando a compressão desta matriz para uma matriz triangular superior. A utilização desta matriz é opcional e pode ser removida.



(a) Topologia.



(b) Árvore.

Índice	0	2	4	6	8			
Linha	B	D	A	C	B	D	A	C
Distância	10	5	10	5	5	10	5	10

(c) CSC.

M	A	B	C	D
A	X	10		5
B	X	X	5	15
C	X	X	X	10
D	X	X	X	X

(d) Matriz.

Figura 7 – Representação da topologia mostrada em (a)-(b) de acordo com as estruturas propostas.

A Figura 7 apresenta as estruturas propostas para representar a topologia da Figura 7a, os PEs então em azul, as máquinas em vermelho e os números nas conexões representam uma latência. A Figura 7b apresenta a estrutura hierárquica de árvore de duas das máquinas desta topologia e a Tabela 7c a representação das distâncias das máquinas com CSC. A estrutura matricial para memoização das distâncias entre máquinas é representada pela Tabela 7d. Ela utiliza uma estrutura matricial proposta para a memoização de distâncias entre as máquinas, onde a primeira linha e primeira coluna indicam uma máquina e as células internas representam as distâncias. Um 'X' indica que a posição não é armazenada para economia de memória e um espaço em branco indica uma distância ainda não calculada mas com memória alocada.

Nesta estrutura, a complexidade de acesso de uma distância qualquer é de $O(g)$ e de $O(1)$ caso tenha sido calculada anteriormente. Sendo c a média do número de PEs por máquinas e m o número de máquinas, o uso de memória é de $O(mc)$ para as estruturas de árvore, de $O(\frac{p+l}{c})$ para o CSC e de $O(\frac{p^2}{2c^2})$ para a matriz de memoização. O custo de criação e inicialização da estrutura é dependente do número de PEs na rede e do tempo de acesso e descobrimento das informações topológicas usadas de base, como a do *hwloc* ou do *runtime* Charm++.

Método	Entrada	Saída
num_pes	-	Número de PEs na topologia
num_machines	-	Número de máquinas na topologia
num_nodes	-	Número de nós na topologia
machine_of	um PE	Máquina onde o PE se encontra
node_of	um PE	Nó onde o PE se encontra
pes_of_machine	uma máquina	Uma lista dos PEs da máquina
on_same_machine	uma PE	Uma lista dos PEs na mesma máquina
pes_of_node	um nó	Uma lista dos PEs do nó
on_same_node	uma PE	Uma lista dos PEs no mesmo nó
net_neighbors	uma máquina	Uma lista de máquinas vizinhas
num_net_links	-	Número de *links* na rede
fill_memoi	-	Preenche a matriz de memoização

Tabela 5 – Métodos de informação de topologia. A primeira coluna apresenta a assinatura do método. A coluna Entrada apresenta as informações requeridas pelo método e a coluna Saída indica o que o método retorna.

3.3 Funcionalidades

A interface da Net Topo contém métodos para obter informações da topologia, cálculo de distância, *hops* e proximidade, abordados na Seção 3.3.1, e para a serialização das estruturas em um arquivo XML, descrito na Seção 3.3.2. Na Seção 3.3.3 é detalhado como são inicializadas as estruturas da abstração. A Tabela 5 apresenta as informações de topologia disponíveis na abstração, seus métodos e suas assinaturas. A palavra "nó" é utilizada para representar um agrupamento interno de PEs de uma máquina, realizado por nível de *cache* compartilhada.

3.3.1 Funções de Distância e Proximidade

Métricas de proximidade entre dois PEs e a distância entre eles são relevantes para realizar uma alocação de tarefas eficiente e que leve em consideração a topologia de rede utilizada. Três métricas de distância foram implementadas dentro da Net Topo: uma função de proximidade, um cálculo de *hops* e um cálculo de distância mínima.

Como os custos de comunicação aumentam conforme a distância entre PEs, saber em que nível a comunicação irá acontecer permite uma estimativa do custo de comunicação entre dois PEs. Para determinar qual a camada de comunicação entre dois PEs, basta saber da topologia de máquina de uma delas. Se não estiverem na mesma máquina, esta acontecerá em rede. Se estiverem, é possível determinar em qual nível. A função de proximidade realiza um cálculo simples para indicar o quão próximo um PE está de outro na topologia de máquina. O método que implementa esta função tem o nome de *proximity*, recebe dois identificadores de PE como entrada e retorna um inteiro de acordo com a função de proximidade descrita na Tabela 6. Saber

o quão próximo um ponto está de outro em uma rede é dependente ou de sua distância mínima ou da quantidade de *hops* até ele, que caem no escopo das próximas funções e não são tão simples.

Saída	Relação entre os dois PEs
3	São o mesmo.
2	Estão na mesma máquina e no mesmo nó.
1	Estão na mesma máquina mas não no mesmo nó.
0	Não estão na mesma máquina.

Tabela 6 – Função de proximidade entre dois PEs. A primeira coluna representa a saída da função e a segunda a relação entre os dois PEs.

Um cálculo de *hops* permite observar a proximidade entre dois PEs em uma rede, a quantidade de *links* utilizada em uma comunicação entre os dois e ter uma noção de distância sem precisar obter as latências da rede. O cálculo de *hops* é implementado pelo método *hop_count*, que recebe identificadores de dois PEs, um origem e um destino, e retorna um inteiro com o número de *hops*. A implementação é uma busca em largura pela estrutura CSC, iniciando no PE origem e contando o número de expansões até alcançar o PE destino. Esta implementação não utiliza o mecanismo de memoização, pois as distâncias mínimas são diferentes do número de *hops*.

Um cálculo de latência mínima entre dois PEs permite considerar com precisão a latência de comunicação entre eles mas não oferece uma perspectiva de contenção. Essa informação é útil para um escalonador ou balanceador que visa reduzir as latências de comunicação do sistema.

O cálculo de distância mínima foi implementando utilizando o algoritmo de Djisktra modificado para incluir o mecanismo de memoização. Antes de ser executado na camada de rede, é obtida a máquina dos dois PEs e é verificado se são os mesmos. Caso sejam, é retornado 0.1 se estão no mesmo nó, 0 se são o mesmo PE e 0.2 caso nenhum dos dois seja verdade.

O Algoritmo 1 apresenta o algoritmo modificado. M é uma função cujo domínio são dois PEs da topologia, x e y, e a imagem é a distância entre eles seguindo uma matriz de distâncias. A função representa o mecanismo de memoização implementado com uma matriz triangular superior. Para inserir um valor na posição (x, y) da matriz de distâncias de M, se utiliza a notação $M[x][y] \leftarrow$ distância. O mapeamento $D : m \rightarrow d$ associa uma máquina m a uma distância d da origem. A notação $D(m) \rightarrow d$ é utilizada para indicar que uma nova associação foi criada.

No inicio do Algoritmo 1 é verificado se a distância entre a origem e o destino já não é conhecida na estrutura de memoização. Caso seja, a distância é retornada e o algoritmo encerra. Nas linhas 1 a 10 ocorre a inicialização do algoritmo, adicionando máquinas cuja distância já

se sabe ao conjunto das não visitadas. Nas linhas 11 a 27 é executado o laço principal, onde, para cada máquina do conjunto de máquinas não visitadas (U), se alcança suas vizinhas e, caso não tenham sido alcançadas, as adiciona em U. Caso a máquina vizinha tenha sido alcançada, é verificado se a distância atual até ela não é menor do que a armazenada no momento (linha 18). Antes de trocar a máquina sendo visitada (linhas 20 a 22), se insere sua distância em M para uso futuro, a adiciona ao conjunto das visitadas (V) e a retira de U. Ao se trocar de máquina (linha 23) é sempre escolhida a máquina menos distante de U e, caso seja o destino, sua distância é adicionada em M e o algoritmo termina retornando a distância encontrada. Caso não se alcance o destino, é retornado -1.

Foi cogitado a importação das distâncias memoizadas de uma máquina vizinha na linha 12 com um laço adicional, mas geraria um sobrecusto que não seria sempre útil. A utilização dela não é sempre melhor devido ao fato de que a distância memoizada é relativa à distância de alcançar a máquina sendo visitada, portanto não necessariamente será o menor até o alvo, mas somente a menor a partir do atual.

3.3.2 Serialização

O armazenamento da estrutura de topologia permite utilizar a mesma estrutura topológica em execuções diferentes, reduzindo o custo de inicialização da abstração e mantendo persistência das informações de custos já calculadas.

Para realizar a serialização das estruturas da Net Topo, foi utilizado a biblioteca de serialização *Cereal* (University of Southern California, 2013). A biblioteca foi escolhida pela sua simplicidade, velocidade e pela compatibilidade com a linguagem *C++*. O arquivo de serialização gerado está na linguagem XML (*Extensible Markup Language*), o mesmo utilizada pelo *hwloc*, pois oferece legibilidade do que está armazenado.

Foram criadas as funções *save_topology* e *load_topology* para exportar e importar, respectivamente, um arquivo XML com nome passado por parâmetro. Caso nenhum nome seja passado, o nome padrão utilizado é "*net_topo.xml*". Para inicializar as estruturas com as informações armazenadas num arquivo, basta chamar a função *load_topology* após a criação de uma instância da Net Topo.

3.3.3 Inicialização

Para ampliar e facilitar seu uso, a Net topo é uma abstração desacoplada de outras implementações, o que faz que sua inicialização seja dependente de alguma fonte de informações topológicas. As informações de rede podem ser obtidas manualmente ou através de mecanismos automáticos, como o *netloc*. O *runtime* Charm++ obtém informações de topologia automaticamente. A Seção 3.3.3.3 apresenta detalhes das informações obtidas pelo Charm++ e como estas informações são passadas para a abstração.

Algoritmo 1: Djisktra Modificado

Entrada: origem: máquina de origem, destino: máquina destino
Saída: Distância entre a origem e o destino
Estruturas: V: Conjunto de máquinas visitadas,
U: Conjunto de máquinas não visitadas,
D: Mapeamento de uma máquina para uma distância.

```
1  U ← {origem}
2  D(origem) ← 0
3  para cada máquina "x"da topologia faça
4      U ← U ∪ {x}
5      se M[origem][x] existe então
6          D(x) → M[origem][x]
7      senão
8          D(x) → ∞
9  atual ← origem                                  // máquina sendo iterada
10 dist ← 0                                        // distância até o atual
11 enquanto U ≠∅ faça
12     para cada máquina "v"| v é vizinha da máquina atual faça
13         dist_v ← dist + distância entre atual e v
14         se v ∉ {V ∪ U} então
15             U ← U ∪ {v}
16             D(v) → dist_v
17         senão
18             se v ∈ U e D(v) > dist_v então
19                 D(v) → dist_v
20     M[origem][ atual] ← dist
21     V ← V ∪ {atual}
22     U ← U \ {atual}
23     atual ← x | x ∈ U e tenha a menor distância em U
24     dist ← D(atual)
25     se atual = destino então
26         M[origem][ destino] ← dist
27         retorna dist
28 retorna −1                                      // não alcançou o destino
```

Neste trabalho foram criadas três inicializações: uma manual, que recebe uma entrada padrão; uma automática, utilizando um adaptador para o Charm++; e uma através de um arquivo XML gerado anteriormente pelo Net Topo. As três formas de inicialização são abordadas a seguir e todas têm a opção de remover o mecanismo de memoização.

3.3.3.1 Inicialização Manual

Um padrão de entrada com as informações de topologia necessárias foi criado para a função de inicialização do Net Topo. O algoritmo de funcionamento da inicialização da estrutura

Algoritmo 2: Inicialização da Topologia de Máquina

Entrada: T: Conjunto ordenado de triplas (m, pe, n), onde m é uma máquina, pe um PE desta máquina e n o nó deste PE.
Saída: PM: Conjunto de duplas (pe, m) onde pe é um PE e m sua máquina,
PN: Conjunto de duplas (pe, n) onde pe é um PE e n seu nó,
MP: Mapeamento de cada máquina ao seu conjunto de PEs,
NP: Mapeamento de cada nó aos seus PEs.
Estruturas: P: Conjunto de PEs de uma máquina.

```
1  maq ← 0, no ← 0                          // máquina e nó sendo iterados
2  n_nos ← 0
3  P ←Ø, PM ←Ø, PN ←Ø
4  para cada tripla (m, pe, n) em T faça
5      PM ← PM ∪ {(pe, m)}
6      se m ≠ maq então
7          MP(maq) → P
8          maq ← m
9          n_nos ←| NP |
10         P ←Ø
11     no ← n + n_nos
12     PN ← PN ∪ {(pe, no)}
13     NP(no) → pe
14     P ← P ∪ {pe}
15 MP(maq) → P
```

foi dividido em dois e é descrito pelos Algoritmos 2 e 3.

A inicialização da topologia de máquina é descrita pelo Algoritmo 2 e deve ser executada antes, por capturar informações necessárias para a divisão de rede. O objetivo deste algoritmo é criar um mapeamento entre PEs, nós e máquinas, para a representação da topologia de máquina. O algoritmo necessita como entrada um conjunto de triplas $(pe, m, n) \mid pe, m, n \in \mathbb{Z}$ onde pe representa um PE, m representa a máquina deste PE e n o seu nó. Os nós recebidos na entrada são relativos a uma máquina e não ao sistema como um todo. O algoritmo assume que todos os PEs de uma máquina estejam em sequência nesta lista. O resultado deste algoritmo são duas listas de duplas PM e PN, um agrupamento de PEs por nó e um agrupamento de PEs por máquina. O mapeamento $MP : m \rightarrow P$ associa um conjunto de PEs P a máquina m. Para associar o conjunto P a uma máquina m no mapeamento MP, é usada a notação $MP(m) \rightarrow P$. O mapeamento $NP : n \rightarrow P$ associa um conjunto de PEs P a um nó n. Para adicionar um PE pe ao conjunto P de um nó n, é utilizado a notação $NP(m) \rightarrow pe$. A notação $\mid NP \mid$ é utilizada para indicar o número de mapeamentos em NP.

Nas linhas 1 e 3 do Algoritmo 2 são feitos os ajustes iniciais para a execução. O laço principal (linhas 4 a 15) mapeia cada PE para a sua máquina e para o seu nó (com os conjuntos PM e PN). Cada máquina e cada nó é associado a um conjunto de PEs com os mapeamentos

Algoritmo 3: Inicialização da Topologia de Rede

Entrada: T: um conjunto de triplas (pe, v, d), onde pe e v são PEs vizinhos e d a distância entre eles.
Saída: G: um grafo de máquinas que representa a topologia de rede.
Estruturas: D: conjunto de duplas $\{v, d\}$, onde v é uma máquina e d uma distância.

```
1 V ← Ø, E ← Ø
2 para cada máquina m na topologia faça
3     D ← vizinhas(m, T)
4     V ← V ∪ {m}
5     para cada dupla (v, d) em D faça
6         V ← V ∪ {v}
7         E ← E ∪ {(m, v)}
8         W(m, v) → d
```

MP e NP, respectivamente. Na linha 11 é re-identificado o nó do PE, para ser absoluto e não relativo à máquina que estava. Quando se troca a máquina sendo iterada (linhas 6 a 10) são agrupados para a máquina todos os PEs em P para utilizar no mapeamento MP. Na troca de máquina também é atualizada a contagem de nós encontrados até o momento, para modificar o número de identificação dos nós.

O algoritmo de inicialização de topologia de rede, Algoritmo 3, captura as informações de vizinhança descrita por uma entrada (T), estabelece as vizinhanças em nível de máquina e as armazena em um grafo G. O grafo G é um grafo representando uma topologia de rede e é dado por $G = (V, E, W) \mid V =$ { união dos v, que representa máquinas}, $E =$ { união das relações $(v, u) \mid v, u \in V$}, W é uma função de peso $W : v, u \rightarrow w$. Este grafo é implementado com um CSC. A função $vizinhas$ utilizada é descrita pelo Algoritmo 4 e encontra as máquinas vizinhas de uma determinada máquina e suas distâncias até ela.

O algoritmo de inicialização da topologia de rede busca todas as máquinas vizinhas de cada máquina na topologia (linha 3). Esta máquina é adicionada ao grafo como um vértice (linha 4) e para cada máquina vizinha é criado uma aresta com peso equivalente a distância até ela (linhas 7 e 8).

O Algoritmo 4 estabelece um conjunto das máquinas vizinhas de m e a distância entre elas. A funcionalidade $maquina\_de(pe)$ retorna a máquina de um PE pe. O laço principal do algoritmo obtém todos os vizinhos de cada PE da máquina, criando duplas de máquinas vizinhas com as suas distâncias, agregadas no conjunto V. Como cada máquina tem vizinhança consigo mesma, é necessário retirar ela mesma do mapeamento (linha 10). Para obter a distância de uma máquina a outra, a menor distância dentre os PEs é utilizada, assumindo que uma latência entre PEs da mesma máquina possa ser desconsiderada no cálculo.

Algoritmo 4: Máquinas Vizinhas

Entrada: T: um conjunto de triplas (pe, v, d), onde pe e v são PEs vizinhos e d a distância entre eles,
m: uma máquina cuja vizinhança se quer conhecer.

Saída: V: conjunto de duplas $\{v, d\}$, onde v é uma máquina vizinha a m e d a distância entre elas.

```
1  maq_v ← 0
2  V ←∅
3  para cada tripla (pe, v, d) em T | maquina_de(pe) = m faça
4      maq_v ← maquina_de(v)
5      se maq_v ∉ V então
6          V ← V ∪ {maq_v, d}
7      senão
8          se V(maq_v) > d então
9              V(maq_v) ← d
10 V ← V \ {m}
```

3.3.3.2 Inicialização via XML

A inicialização XML importa informações salvas anteriormente em um arquivo XML, restaurando a topologia armazenada e sua matriz de distâncias. Para utilizar esta inicialização basta invocar o método *load_topology*, descrito na Seção 3.3.2. A utilidade desta inicialização se dá pela remoção do tempo de descobrimento da topologia e pela importação da matriz de distâncias utilizada em execuções anteriores. Esta inicialização só pode ocorrer se houver um arquivo XML de topologia salvo anteriormente.

3.3.3.3 Charm++

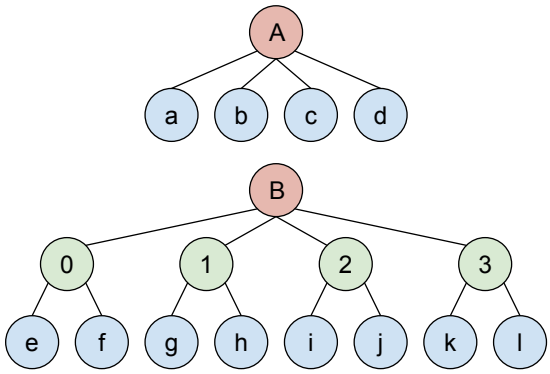
A inicialização através do *runtime* do Charm++ é automática mas apresenta uma série de limitações devido a incompletude das informações coletadas pelo *runtime* e pelo armazenamento destas, abordadas adiante.

O Charm++ possui duas maneiras de descobrir uma topologia: de maneira manual, com opções de execução, da classe *TopoManager* e topologias fixas como *Cray* e *BlueGene*; e de maneira automática, através de envio de mensagens entre os PEs utilizados e enquadramento da topologia encontrada em um dos tipos pré-estabelecidos em sua classe *topology*, como *Mesh* 2D, *Torus* ou *Fat-tree*. A escolha destas abordagens se deve a otimizar o calculo de *hops*, que depende só das posições da origem e do destino nestas topologias, evitando um algoritmo com travessia de grafo. Porém, os únicos métodos obrigatórios em cada topologia são os de encontrar o número máximo de vizinhos e os seus vizinhos, o que acarreta em dificuldades para algoritmos que consideram topologias genéricas.

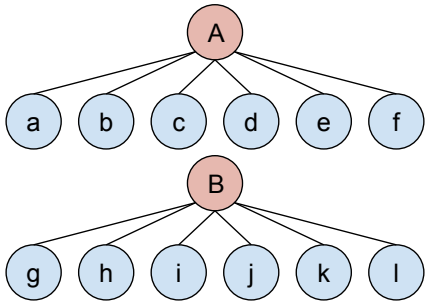
A obtenção de topologia do Charm++ inclui três problemas que não eram esperadas no

inicio deste trabalho: (i) não há armazenamento de latência de mensagens entre PEs reconhecidos na topologia; (ii) não há informações sobre compartilhamento de *cache* entre PEs; e (iii) há imprecisões em certas funções quando as máquinas da rede não são iguais. A ausência de latências impossibilita cálculos de distância que não sejam através de *hops*, pois não há nenhuma outra métrica. A falta de informações sobre compartilhamento de *cache* impede o agrupamento de PEs por grupos de *cache* e um detalhamento de cada máquina na topologia. As funções imprecisas são relacionadas ao número de PEs por nó físico da topologia, que são distribuídos igualmente pelo número de máquinas na rede. Esse agrupamento equivalente impossibilita saber quais PEs estão efetivamente em quais máquinas, gerando informações imprecisas para a secção entre topologia das máquinas e topologia da rede.

A Figura 8 apresenta uma comparação entre uma topologia real com máquinas heterogêneas e a sua representação dentro do sistema do Charm++. A topologia é composta por duas máquinas, uma com 4 cores, sem agrupamento de *cache*, e a segunda com 8 cores e 4 agrupamentos de *cache*.



(a) Topologia Real



(b) Agrupamento realizado pelo Charm++

Figura 8 – Comparação entre a topologia real (esquerda) e a encontrada pelo Charm++ (direita). Máquinas em vermelho e letras maiúsculas, agrupamentos de *cache* em verde e numerais, e PEs em azul e letras minúsculas.

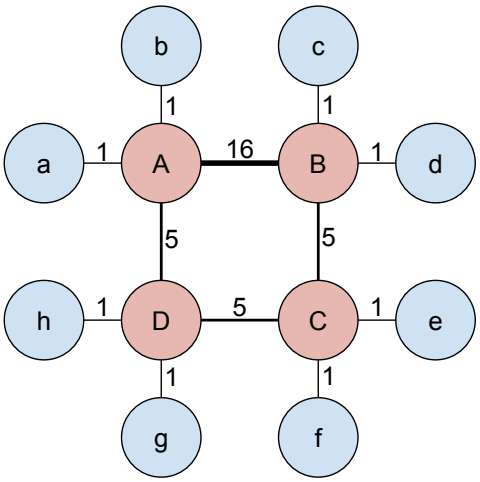
Para interagir com o *runtime* do Charm++ e extrair as informações topológicas foi criado um adaptador de nome *net_topo_charm_proxy*. Dentro do adaptador foram criados duas inicializações intermediárias: *init* e *init_no_nodes*. O primeiro extrai os agrupamentos de máquinas enquanto o segundo ignora a parte de topologia de máquina do Charm++ e trata cada PE como uma máquina individual. Portanto, o segundo mantém a precisão das conexões mas perde qualquer informação relacionada ao agrupamento de PEs em máquinas. Ambas as inicializações utilizam o padrão de inicialização da Net Topo.

4 Experimentos

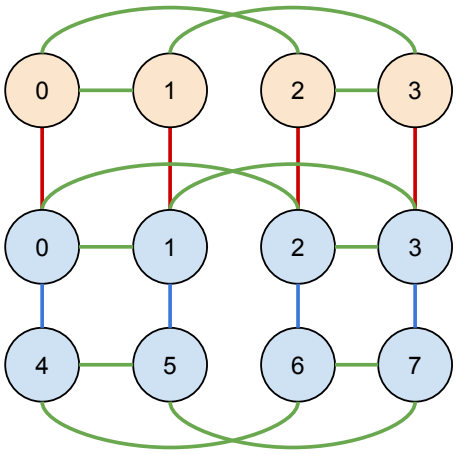
Neste capítulo são abordados os testes e experimentos realizados, visando demonstrar o funcionamento da abstração Net Topo e avaliar sua escalabilidade. A Seção 4.1 apresenta a metodologia de testes utilizada e as funcionalidades cobertas pelos testes. Na Seção 4.2 é descrito um balanceador de carga e a sua modificação, utilizados para comparação. Na Seção 4.3 é observado o sobrecusto da abstração comparado um balanceador com a abstração nativa do Charm++ com o mesmo balanceador mas com a Net Topo. Por último, são apresentadas os tempo de execução de algumas das funções da abstração e a escalabilidade delas na Seção 4.4.

4.1 Testes de Funcionamento

A comprovação do funcionamento da Net Topo foi feito com a utilização de testes de unidade sob topologias manualmente inicializadas. Foram implementados 132 testes para cobrir o funcionamento adequado dos métodos e seus componentes, garantindo seu funcionamento independente de sua forma de inicialização. Foi confeccionado um teste para cada tipo de modificação e retorno previstos nas funcionalidades, como valores nos limites da memória. Por exemplo, na inserção da estrutura de matriz reflexiva e sem diagonais, foi testado uma inserção comum, uma em cada extremidade da matriz, uma em diagonal, uma reflexiva e uma inválida.



(a) Topologia simétrica com um *link* custoso.



(b) Topologia assimétrica. *Links* azuis, verdes e vermelhas têm custo 1, 2 e 8, respectivamente.

Figura 9 – Topologias utilizadas para teste.

A Figura 9 apresenta duas topologias utilizadas para testar as funcionalidades da abstração. Como a inicialização das topologias foi manual, não foram utilizadas topologias muito grandes. A topologia (a) tem como objetivo verificar o funcionamento correto dos métodos de distância e *hops*. A topologia (b) apresenta um ambiente heterogêneo e um maior detalhamento

da estrutura interna de cada máquina, permitindo observar a divisão adequada entre topologia de máquina e de rede feita pela abstração. Para avaliar um funcionamento mais extenso na topologia de rede, a topologia (b) é utilizada novamente considerando cada PE como uma máquina individual.

As funcionalidades cobertas pelos testes são: (1) número de PEs na topologia; (2) número de máquinas na topologia; (3) PEs de uma máquina; (4) máquina de um determinado PE; (5) nó de um determinado PE; (6) outros PEs na máquina de um determinado PE; (7) PEs em uma determinada máquina; (8) outros PEs no mesmo nó de um determinado PE; (9) PEs em um determinado nó; (10) número de *links* na camada de rede; (11) nome da topologia; (12) mudança de nome da topologia; (13) vizinhos na camada de rede; (14) proximidade entre dois PEs; (15) distância mínima entre dois PEs; (16) *hops* entre dois PEs; (17) serialização de uma topologia; (18) leitura de uma topologia; (19) inicialização de uma topologia; e (20) persistência de informações entre execuções.

Em relação às estruturas utilizadas foram cobertas as seguintes funcionalidades: (1) criação de um CSC com um mecanismo de memoização; (2) inserção em um CSC; serialização de um CSC; (3) cálculo de distância mínima em um CSC; (4) utilização da memoização no cálculo de distâncias do CSC; (5) criação de uma matriz triangular superior sem diagonal; (6) inserção em uma matriz triangular superior sem diagonal; e (7) serialização de uma matriz triangular superior sem diagonal.

4.2 Balanceador de Carga

Para a demonstração do funcionamento da abstração da topologia de rede, foi modificado um balanceador de carga distribuído presente na plataforma Charm++, de nome NeighborLB. A modificação realizada só alterou de onde as informações de topologia foram retiradas, substituindo a implementação nativa pela Net Topo, sendo que o algoritmo do balanceador não foi alterado. Este balanceador observa somente PEs vizinhos para a tomada de decisão de balanceamento, evitando contenção da rede e obtendo baixa latência nas mensagens. Assim como outros balanceadores distribuídos, o NeighborLB é escalável, pois independe do número de nós da rede. Porém, seu balanceamento é limitado pela quantidade de informação.

O descobrimento de vizinhos no balanceador NeighborLB é dado através de um balanceador base que existe dentro da plataforma Charm++ que utiliza seu mecanismo de inferência. Este mecanismo foi substituído pela abstração criada para comprovar seu funcionamento.

A escolha do balanceador NeighborLB se deu pela sua simplicidade e por observar somente seus vizinhos, criando um cenário de comparação equivalente para a abstração Net Topo. Os métodos de cálculo de *hops* do Charm++ são mais velozes por considerarem topologias especificas (e as vezes imprecisas), o que causaria um cenário onde não se poderia avaliar o sobrecusto da abstração de maneira equivalente. Nenhum dos balanceadores que utiliza as in-

formações topológicas do Charm++ possui um cálculo de distância, por não possuir nenhuma informação de latência, inviabilizando a comparação neste nível.

4.3 Avaliação de Sobrecusto

Para observar o sobrecusto da abstração, foi comparada a execução do balanceador NeighborLB, detalhado na Seção 4.2, com o balanceador myNeighborLB, uma modificação do anterior que utiliza a Net Topo em vez do sistema topológico do Charm++. O objetivo da comparação é observar se há uma discrepância de desempenho no uso da abstração, verificar o uso funcional das abstração junto ao modulo de balanceamento de carga do Charm++ e observar possíveis ganhos e perdas de desempenho da abstração.

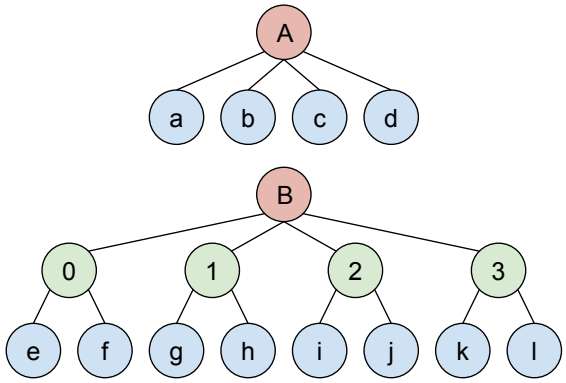
O *benchmark* sintético *LB Test* foi utilizado para realizar uma avaliação da execução dos balanceadores de carga. Nele são criadas cargas uniformemente distribuídas entre uma valor de carga mínimo e máximo. O *LB test* cria um perfil de comunicação entre estas cargas e invoca um balanceador de carga periodicamente, ambos escolhidos pelo usuário. As configurações do *LB test* são abordadas na Subseção 4.3.2. O ambiente de execução do *benchmark* é determinado por um arquivo de grupos de execução, contendo as máquinas que serão utilizadas e seus IPs. No caso deste trabalho, as máquinas Ciclope e Centauro foram utilizadas, detalhadas a seguir.

4.3.1 Ambiente de Execução

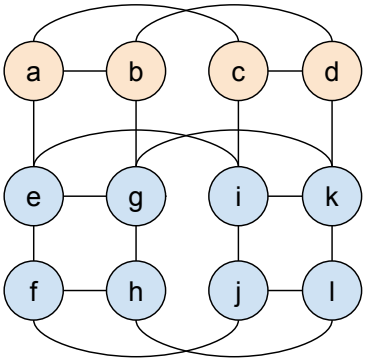
A execução foi realizada em um conjunto de duas máquinas diferentes: (i) Ciclope e (ii) Centauro. A primeira máquina utiliza 4 CPUs Intel Core i7-7700 @3.60GHZ com *hyper-treading* e possui 8GB@1200 MHz de memória. Possui 3 níveis de cache com tamanhos de 32 KB, 256 KB e 8 MB. Sua versão de GCC é 5.5.0. A segunda máquina possui 4 CPUs Intel Core i5-7400 @3.00GHz e 16GB@2400 MHz DIMM de memória. Possui 3 níveis de cache com tamanhos de 32 KB, 256 KB e 6 MB. GCC instalado na versão 5.4.0. Ambas as máquinas utilizam Linux Mint na versão 18.2 e Charm++ 6.9.0. A Figura 10 apresenta a topologia de cada máquina e do ambiente como um todo.

4.3.2 Casos de Execução

Foram realizados três configurações do *LB test* para realizar os experimentos capturados neste trabalho, com os seguintes nomes: (i) *init*, (ii) *LB test M* e (iii) *LB test G*. A Tabela 7 apresenta o nome de cada configuração e o número de execuções de cada uma delas. A primeira configuração visa observar os tempos de inicialização da abstração de topologia Net Topo, do armazenamento dela, da inicialização da topologia nativa do Charm++ e da inicialização através de um arquivo XML. Portanto, não é preciso executar nenhum balanceamento de carga e o *LB test* foi utilizado por conveniência. As configurações (ii) e (iii) visam observar variação no desempenho do balanceador. Em ambas foi utilizada uma execução de 150 iterações com o



(a) Em vermelho os agrupamento das máquinas Centauro (A) e Ciclope (B). Em verde os agrupamentos de *cache*.



(b) Topologia de rede do ambiente. Círculos azuis representam os PEs da máquina Ciclope e em laranjas os da máquina Centauro.

Figura 10 – Topologia do ambiente de execução. Os agrupamentos de cada máquina estão em (a) e a topologia de rede em (b).

balanceador sendo invocado a cada 40 iterações, o intervalo de carga foi de 30 a 4120ms e o padrão de comunicação utilizado foi de anel. A diferença entre as configurações (ii) e (iii) é o número de tarefas criadas. Na configuração *LB test M* foram criadas 3600 tarefas (300 tarefas por PE) enquanto na configuração *LB test G* este número foi aumentado para 5400 (450 tarefas por PE). A execução de dois níveis de carga diferentes visa observar alguma mudança devido ao Net Topo e relativa a carga, que não deveria ocorrer.

Por fim, as três configurações foram executadas no ambiente Ciclope + Centauro com a opção de execução "++*LBtestPESpeed*" que permite que balanceadores coletem e utilizem a velocidade das CPUs de cada máquina para o balanceamento.

Nome	**Número de execuções**
Init	120
LB test M	20
LB test G	15

Tabela 7 – Nome das configurações e seu número de execuções.

4.3.3 Resultados

Os resultados foram divididos em (i) inicialização (ii) e balanceamento de carga. Na primeira parte (inicialização) é avaliada a configuração *init*, observando os custos de inicialização de estruturas para a execução. Na segunda parte (balanceamento de carga) é observado os sobrecustos da utilização da Net Topo utilizando as configurações *LB test M* e *LB test G* com os balanceadores NeighborLB (NLB) e myNeighborLB (myNLB).

4.3.3.1 Inicialização

Na configuração de *init* foram observadas quatro inicializações: (i) a Nativa do Charm++, (ii) a do Net Topo, (iii) a integração destas duas e (iv) a inicialização Net Topo com um arquivo XML. A inicialização (i) realiza uma série de mensagens IP entre as máquinas, encontrando e identificando cada PE de cada máquina as enquadrando para uma das topologia internas. A (ii) coleta as informações da topologia do Charm++, as organiza para a estrutura do Net Topo e as armazena, salvando tudo em um arquivo XML. Como esta inicialização é dependente da nativa, a soma das duas também foi calculada como a inicialização (iii). Por último, a importação XML (iv) captura as informações armazenadas por uma inicialização anterior do Net Topo.

Inicialização	**Tempo médio de execução (ms)**	**Desvio Padrão (ms)**
Nativa do Charm++	24,46	5,22
Net Topo	0,87	0,18
Nativa + Net Topo	25,33	5,41
Importação XML	0,72	0,06

Tabela 8 – Inicializações testadas, seus tempos de execução e seu desvio padrão.

A Tabela 8 apresenta o tempo e desvio padrão de cada uma das inicializações observadas. Quando comparada com a nativa, a adição da inicialização da Net Topo gerou em média 3.5% de sobrecusto. É importante ressaltar que apesar da inicialização da Net Topo ter tido uma grande variância, quando somada a execução nativa não aumentou o desvio padrão. Quando comparada a nativa, a importação XML reduz em média o tempo de inicialização em 97.1%, agilizando o processo de inicialização do sistema. Conforme o número de PEs do sistema aumenta, é esperado que este ganho entre a importação e a inicialização cresça ainda mais, pois a implementação nativa realiza uma multitude de mensagens IPs para encontrar e definir a topologia. Por outro lado, o custo de inicialização da Net Topo deve crescer junto com o número de PEs. Não se sabe se o sobrecusto dela irá se manter, aumentar ou reduzir quando adicionada a nativa, mas se o desempenho da importação se mantiver, é provável que o sobrecusto de inicialização do Net Topo se pague na segunda execução.

4.3.3.2 Balanceamento de Carga

Nas configurações *LB test M* e *LB test G* foram capturadas duas informações para comparação: (i) o tempo de balanceamento e (ii) o tempo de execução da aplicação. A primeira informação permite observar modificações na velocidade do balanceador, a análise deste busca observar o sobrecusto da abstração. A segunda avalia mudanças no balanceamento da aplicação, que não deveria ser afetado, pois não há mudanças no comportamento do balanceador.

Na Tabela 9 é observado o tempo e o desvio padrão de balanceamento de NLB e de myNLB nas configurações de teste *LB test M* e *LB test G*. Ambos as configurações apresentaram

uma melhora na abordagem com Net Topo, com uma redução de tempo de 0.18 ms e 0.19 ms, respectivamente. Como esta pequena melhoria não escalou, significa que não é dependente da quantidade de carga, a diferença entre as configurações. Essa melhoria indica que a abstração criada oferece um acesso a vizinhos ligeiramente mais rápida que a nativa do Charm++, pois ambas as execuções e realizam o mesmo número de invocações para encontrar vizinhos.

Configuração	*LB test M*		*LB test G*	
	NLB	*myNLB*	*NLB*	*myNLB*
Tempo médio de balanceamento (ms)	19,60	19,42	29,12	28,93
Desvio Padrão (ms)	2,85	2,78	4,22	3,83

Tabela 9 – Tempo médio de balanceamento de carga e desvio padrão dos balanceadores NeighborLB e myNeighborLB nos casos de execução *LB test M* e *LB test G*.

Os resultados de tempo dos balanceadores de carga são observados no gráfico da Figura 11. Surpreendentemente, a versão modificada obteve em média uma pequena melhoria no tempo total de execução (de aproximadamente 1,5%). As melhorias no tempo de balanceamento não geram tanto impacto (totalizam menos de 1ms no total) e o tempo de inicialização não é considerado no tempo da aplicação. O motivo atribuído a melhoria é a natureza variante do *benchmark* acoplado com número baixo de execuções e um desvio padrão próximo a 10% nas execuções. O que se conclui aqui é que não houve mudanças significativas no comportamento do balanceador.

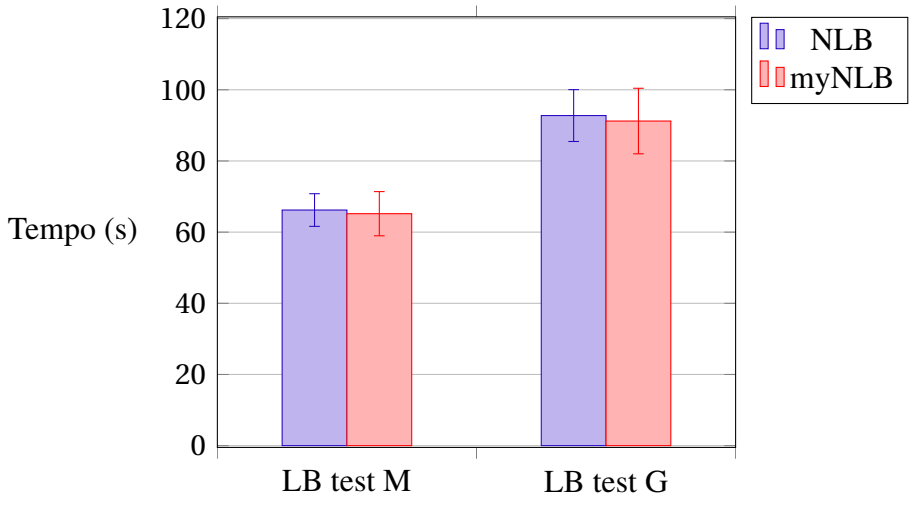


Figura 11 – Comparação de tempos de balanceamento entre NeighborLB e myNeighborLB. Os intervalos em cada barra representam o desvio padrão.

As execuções realizadas com o *benchmark LB test*, o balanceador NeighborLB e sua modificação para utilizar a Net Topo permitiram observar: (i) um sobrecusto de 3,5% na inicialização do Charm++, compensado largamente pelo ganho significativo na inicialização com XML; (ii) um sobrecusto abaixo do esperado no balanceamento, inclusive com menos custo que

o do próprio Charm++; e (iii) não houve um impacto negativo na qualidade do balanceamento, já que o algoritmo não foi modificado. Estes resultados permitem afirmar o funcionamento desejado pela abstração. Por outro lado, os experimentos realizados são bem limitados, pois observam uma escala reduzida de apenas duas máquinas e 12 PEs, o que inviabiliza assunções seguras em relação ao sobrecusto para outras escalas de aplicação. Outra limitação dos experimentos é a avaliação de somente a função de vizinhança da abstração, não podendo, por exemplo, indicar o sobrecusto de se considerar uma topologia genérica em um cálculo de *hops*.

4.4 Avaliação de Desempenho

Para observar o funcionamento da abstração em escalas diferentes, foi criado um gerador de topologias *fat-tree*, *mesh2D* e *mesh3D* e observado os tempos de execução de 4 topologias geradas. Estes casos são apresentados na Seção 4.4.1. A Tabela 10 apresenta as funcionalidades avaliadas e o número de execuções de cada uma. A função de inicialização é representada por *init*, as de arquivamento e importação XML por *save* e *load*, respectivamente, e a de preenchimento do mecanismo de memoização por *fill*. Os cálculos de proximidade são detalhados logo antes de seus respectivos experimentos.

Métodos	**Número de execuções**
init, *save* e *fill*	25
load	100
cálculos de proximidade	200

Tabela 10 – Número de execuções de cada função testada. Um dos cálculos de proximidade foi executado 100 vezes em vez de 200.

A execução dos testes foi realizada na máquina Centauro, apresentada na Seção 4.3.1, utilizando um único núcleo e medindo o tempo através da biblioteca *crono* de *C++*. Não foi utilizado um ambiente maior ou paralelo, pois o objetivo da execução era observar a variação de desempenho da estrutura em escalas diferentes.

4.4.1 Casos de Execução

Foram criados 4 casos de representações de topologias para a execução dos testes: uma *fat-tree* com 7 níveis, nomeada de *tree M*; uma *fat-tree* de 10 níveis, representada como *tree G*; uma *mesh2D* de tamanho 16x16, representada por *mesh2D*; e uma *mesh3D* simétrica de dimensão 8, nomeada *mesh3D*. Todas as topologias tiveram uma versão da abstração com memoização e outra sem. O número de máquinas e ligações de cada um dos casos de teste estão apresentados na Tabela 11. A coluna tamanho do arquivo XML e XML- representa o espaço em disco do arquivo XML gerado pela Net Topo com e sem memoização, respectivamente. É

possível observar que este tamanho não escala bem devido ao dispositivo de memoização, por ocupar espaço com complexidade $O(m^2)$ onde m é o número de maquinas.

Caso	Número de máquinas	Número de ligações	XML/M	XML
tree M	127	126	0,32	0,05
mesh2D	256	538	1,28	0,12
mesh3D	512	1.344	5,06	0,31
tree G	1.023	1.022	20,06	0,42

Tabela 11 – Tamanho das abstrações criadas em número de máquinas, em número de ligações e em ocupação do arquivo XML em MB com e sem memoização, representado por XML/M e XML, respectivamente.

4.4.2 Resultados

Os resultados dos experimentos foram divididos em duas partes: (i) inicialização e serialização e (ii) cálculos de proximidade. A primeira parte contempla os tempos de inicialização, arquivamento e importação XML de cada um dos casos. A segunda compara os tempos de cálculo de proximidade, *hops* e distância de cada um dos casos e do preenchimento do mecanismo de memoização nos casos que possuem memoização. Para reduzir o desvio padrão e apresentar um resultado mais coerente, foram retirados da amostra casos discrepantes que ocorreram com baixa frequência (1% ou menos) pois alteravam a média e o desvio padrão de maneira significante por serem valores de duas ordens de grandeza a mais.

4.4.2.1 Inicialização e Serialização

A Figura 12 apresenta o crescimento do tempo médio de inicialização, arquivamento e importação XML de cada um dos casos com o crescimento de número de máquinas. As topologias *tree M*, *mesh2D*, *mesh3D* e *tree G* são representadas por 2^7, 2^8, 2^9 e 2^{10} máquinas, respectivamente. É possível observar que, quando sem memoização, os tempos de arquivar e importar crescem linearmente ou sublinearmente com a quantidade de máquinas. Por outro lado é fácil observar que o mecanismo de memoização em forma de matriz não escala bem com o tamanho da topologia devido a sua natureza quadrática. Isso se deve pelo fator da matriz de memoização representar metade de todas a ligações possíveis, que são mais de 524 mil no caso *tree G*.

4.4.2.2 Cálculos de Proximidade

A função de proximidade é um cálculo que avalia a proximidade entre dois PEs, observando se estão na mesma máquina ou não. O tempo médio de um cálculo de proximidade foi entre 0,20 e 0,21 microssegundos em todos os casos de execução. O desvio padrão foi entre

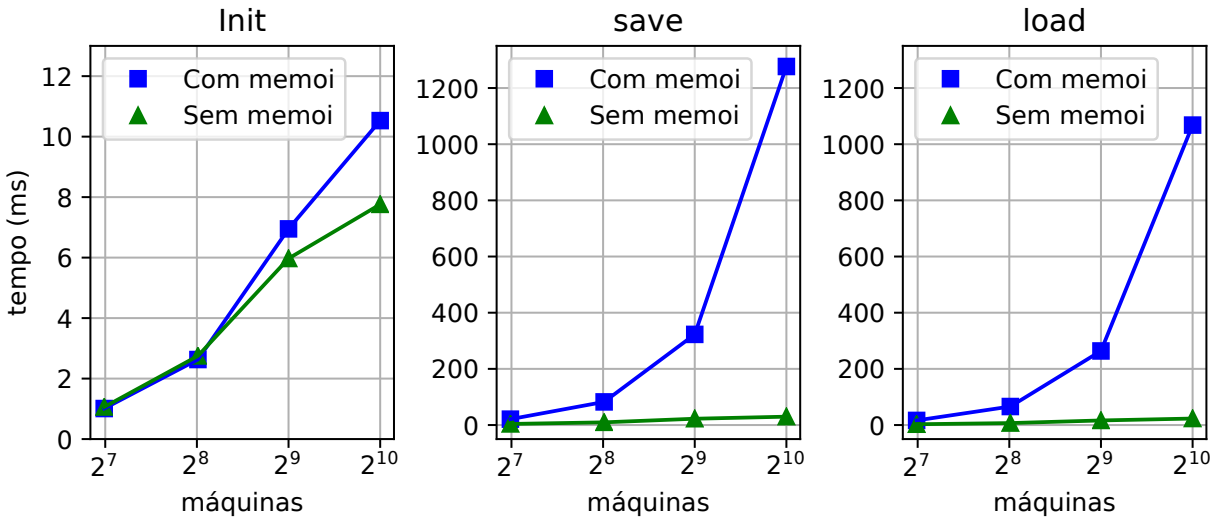


Figura 12 – Crescimento dos tempo médio de inicialização (init), arquivamento (save) e importação XML (load) de cada um dos casos de teste. Os quadrados em azul são os casos com memoização e os triângulos em verde os casos sem memoização.

0,04 e 0,06 microssegundos em todos os casos. Foi observado que o cálculo de proximidade independe do tamanho da topologia e de ter ou não memoização, como era esperado.

O cálculo de distância avalia a latência entre duas máquinas da topologia. Para avaliar a latência deste cálculo, foi agrupado o tempo de todos os cálculos de distância entre uma máquina e todas as máquinas a até dois *hops* de distância dela. Este conjunto de distâncias calculadas foi denominado como "distância II".

O cálculo da distância II utiliza o mecanismo de memoização, e foram avaliados três cenários relativos a memoização: (i) inicialmente completa, (ii) inicialmente vazia e (iii) sem nenhum tipo de memoização. A primeira contêm todas as informações de distância calculadas previamente. A segunda inicia cada cálculo com a memoização inicialmente vazia, expandindo durante a execução de uma distância II. O terceiro caso não utiliza nenhuma memoização e é utilizada para comparação.

A Tabela 12 apresenta os tempos médios de execução do preenchimento da matriz de memoização e dos três cenários de distância II. A primeira coluna apresenta o tempo de preenchimento, que é o tempo o mecanismo de memoização através de cálculos de distância entre todos os nós. A primeira linha de cada caso de execução indica a média dos tempos e a segunda linha o desvio padrão.

Todos os cálculos de distância têm um desvio padrão aumentado devido a variação do grau de cada máquina, sendo de 1 a 3 em uma *fat-tree*, 2 a 4 numa *mesh2D* e 3 a 6 em uma *mesh3D*. O desvio padrão maior no caso de memoização inicialmente vazia se deve ao fator dos cálculos oscilarem entre encontrar um valor armazenado e encontrar um valor novo.

Como o número de distâncias possíveis cresce quadraticamente com o número de má-

Caso		**Preenchimento** (ms)	**Completa** (μs)	**Vazia** (μs)	**Sem memoização** (μs)
tree M	$\overline{x}$	75,70	1,30	11,32	50,56
	σ	0,98	0,04	4,68	13,15
mesh2D	$\overline{x}$	863,91	2,69	75,72	210,42
	σ	5,70	0,04	24,04	41,77
mesh3D	$\overline{x}$	4.880,94	4,96	214,90	873,95
	σ	19,03	0,35	21,12	77,39
tree G	$\overline{x}$	6.215,44	1,40	16,63	39,49
	σ	24,24	0,06	5,74	9,63

Tabela 12 – O tempo médio ($\overline{x}$) e o desvio padrão (σ) de preenchimento completo da matriz de memoização (primeira coluna), e tempo do cálculo de distância II de uma máquina com memoização completa (coluna II), inicialmente vazia (coluna III) ou inexistente (última coluna).

quinas, o tempo de preenchimento também cresce quadraticamente. Este crescimento é observável na Tabela 12. Para que este tempo de preenchimento acelere a execução, é necessário que seu tempo seja pago por um número grande de cálculos de distância. No caso *tree M*, por exemplo, são necessárias 6180 execuções, enquanto para *tree G* são necessárias 125845 execuções.

Quando comparado o tempo de preenchimento da *mesh3D* com da *tree G*, este não cresceu tanto quando comparado com o crescimento entre *mesh2D* e *mesh3D*, por realizar um cálculo de distância com um grau médio menor. Portanto o tempo de preenchimento da memoização cresce tanto com o número de máquinas na topologia quanto com o tempo de um cálculo de distância. Com isso, a quantidade de execuções de distância para que o preenchimento valha a pela varia para cada topologia.

É possível observar um aumento no tempo conforme o grau da topologia aumenta, assim como uma melhoria entre a completa para a inicialmente vazia e outra melhoria entre a inicialmente vazia e a sem memoização. A melhoria no caso da inicialmente vazia se deve ao algoritmo de distância armazenar a distância de todos os visitados, conseguindo encontrar a distância de uma parte da vizinhança nas primeiras execuções e utilizá-la nas seguintes.

Na Tabela 13 são apresentados os tempos médios de cada cálculo de distância individualmente feito na distância II, em vez de um agrupamento destes. O desvio padrão se encontra altíssimo devido a vários fatores, como a variância do grau na topologia, a oscilação entre com e sem memoização e a imprecisão de relógio devido a magnitude sendo avaliada. Devido a esta alta imprecisão os valores são estatisticamente equivalentes e por isso foi avaliado o tempo de vizinhanças, que não tinha tanta variância. Ainda assim, é possível constatar um pequeno crescimento no cálculo de cada distância com o crescimento do grau.

O cálculo de *hops* é outra métrica para avaliar proximidade entre duas máquinas, observando o número mínimo de *links* entre uma máquina e outra. Assim como o cálculo de

Caso		Completa	Vazia	Sem memoização
tree M	$\overline{x}$	0,24	2,05	9,16
	σ	0,03	4,78	13,15
mesh2D	$\overline{x}$	0,25	6,94	19,28
	σ	0,04	14,05	21,74
mesh3D	$\overline{x}$	0,26	11,62	45,82
	σ	0,35	23,11	27,66
tree G	$\overline{x}$	0,27	3,2	7,71
	σ	0,06	5,7	5,6

Tabela 13 – O tempo médio ($\overline{x}$) e desvio padrão (σ) de um cálculo de distância com memoização completa, inicialmente vazia ou inexistente.

distância, foi realizado um cálculo de *hops* entre uma máquina e todas as máquinas a até dois *hops* de distância, agrupando seus tempos num valor denominado de *hops* II. Esse agrupamento foi utilizado para estabilizar os valores, sendo que é possível encontrar a segunda vizinhança com *hops* de maneira muito mais ágil.

Os tempos de *hops* II são apresentados na Tabela 14, comparados com o tempo de distância II. Assim como o cálculo de distância, os cálculo de *hops* demonstrou um aumento no tempo de execução devido ao aumento do grau médio. Não era esperado que os tempo de execução variasse com o número de máquinas, como ocorreu com a *tree G*.

Caso		**Hops** (μs)	**Distância** (μs)	**Sem memoização** (μs)
tree M	$\overline{x}$	23,50	11,32	50,56
	σ	6,36	4,68	13,15
mesh2D	$\overline{x}$	85,11	75,72	210,42
	σ	13,39	24,04	41,77
mesh3D	$\overline{x}$	534,50	214,90	873,95
	σ	7,53	21,12	77,39
tree G	$\overline{x}$	39,23	16,63	39,49
	σ	2,46	5,74	9,63

Tabela 14 – Comparação do tempo médio ($\overline{x}$) e desvio padrão (σ) de *hops* II e distância II.

Era esperado que o cálculo de *hops* fosse mais rápido que o cálculo de distância em todos os casos mas, como não foi utilizado memoização para *hops*, alguns casos de distância foram mais rápidos quando o grau médio foi elevado. Isso se dá pelo armazenamento da distância de múltiplos vizinhos em um único cálculo de distância. É possível observar que houve um acréscimo no tempo do cálculo de *hops*, provavelmente devido a uma redução de localidade da cache.

Por fim, a Tabela 15 apresenta a aceleração do mecanismo de memoização na serialização e na distância II com memoização completa ou inicialmente vazia. É importante que, mesmo com uma aceleração grande na distância, o tempo de preenchimento da memoização pode fazer com que não valha a pena. No caso da *tree G*, por exemplo, seria necessário em torno 164 mil execuções de distância II para que a redução no tempo comece a ser um benefício.

Caso	***save***	***load***	**Vazia**	**Completa**
tree M	0,172	0,185	4,46	38,89
mesh2D	0,105	0,116	2,77	78,22
mesh3D	0,061	0,069	4,06	176,00
tree G	0,021	0,023	2,37	28,21

Tabela 15 – Aceleração geradas pela memoização como matriz superior. As colunas *save* e *load* apresentam os atrasos no tempo médio de arquivamento e importação, respectivamente. As colunas *Vazia* e *Completa* apresentam a aceleração no tempo médio de distância II com memoização inicialmente vazia e completa, respectivamente.

Estes experimentos avaliaram o funcionamento da Net Topo em escalas diferentes e o impacto da memoização implementada. Foi possível determinar que a memoização em forma de matriz não é escalável, causa um detrimento considerável no tempo para o arquivamento e importação e para que seu preenchimento valha a pena, seria necessário um número muito alto de execuções. Apesar disso, ela gera uma aceleração considerável na execução do cálculo de distâncias de uma vizinhança, mesmo quando inicialmente vazia, mas este benefício depende do grau médio da topologia. Em relação a escalabilidade da abstração, os tempos de inicialização, serialização, e proximidade escalaram de maneira adequada, mas os cálculos de distância e *hops* cresceram ligeiramente com o número de máquinas, provavelmente devido a ocupação de memória.

5 Conclusão

Uma distribuição de carga ciente de topologia pode melhorar o desempenho de uma aplicação por considerar latências de comunicação e efeitos de contenção. A obtenção de informações de topologia se dá através da coleta de informações dos sistemas em execução e troca de mensagens. As ferramentas para obtenção destas informações têm limitações e inadequações para o contexto de escalonamento e balanceamento de carga, dificultando seu uso.

Neste trabalho foi desenvolvida a Net Topo, uma abstração de topologia que visa recolher e dispor informações da topologia de rede para uso em escalonamento e balanceamento de aplicações paralelas e distribuídas. Dentre diversas opções de estruturas de dados e representação de topologia, foi utilizado um CSC, uma matriz de distâncias opcional e uma lista de árvores. A abstração criada apresenta diversas informações de topologia e fornece funções de proximidade e distâncias, visando auxiliar o processo de distribuição de carga. A Net Topo também possui diversos mecanismos para sua inicialização, incluindo implementações manuais e importação de um aquivo de topologia em XML.

Foi criado um adaptador para receber informações de inicialização e fazer uma ponte entre a abstração e o *runtime* do Charm++, permitindo utilizar a Net Topo junto ao módulo de balanceamento de carga presente na plataforma. Para comprovar a integração e avaliar o sobrecusto da abstração, foi comparado um balanceador de carga com sua versão modificado para utilizar a Net Topo. Apesar das execuções serem realizadas em baixa escala e em um ambiente pequeno, elas permitiram observar um sobrecusto de inicialização altamente compensado pelo arquivamento da topologia e pouca ou nenhuma mudança negativa nos tempos de execução do balanceador. Foram realizados testes unitários para comprovar o uso de funcionalidades que não foram abrangidas no balanceador, de maneira que possam ser utilizadas independente da forma de inicialização da estrutura.

Por fim, foi implementado um gerador de topologias, avaliado o impacto da memoização e observado a escalabilidade da abstração. O dispositivo de memoização através de uma matriz completa não se mostrou escalável, gerando sobrecustos muito altos conforme o tamanho da topologia cresce. A função de proximidade se apresentou como constante e o restante das funções testadas demonstraram crescimento linear ou sublinear com o aumento no número de máquinas.

5.1 Trabalhos Futuros

A execução deste trabalho criou uma série de oportunidades de trabalhos futuros. Segue uma lista de extensões, melhorias e avaliações cogitadas:

- Realizar uma distribuição das estruturas de topologia de modo que não seja puramente replicada em cada nó e reduza o espaço utilizado em cada máquina, mas ainda mantenha a possibilidade de uma travessia de grafo para cálculo de *hops* ou latência.
- Modificar as estruturas da abstração para serem dinâmicas, possibilitando modificações e atualizações em uma topologia prévia, sem ter de reinicializá-la.
- Implementar uma inicialização utilizando informações do *hwloc* para a topologia de máquina e o Charm++ para a topologia de rede, retirando as imprecisões relacionadas a máquinas do Charm++ enquanto se mantém a automatização do processo.
- Ampliar o gerador de topologias para a abstração, oferecendo casos de teste para uma gama maior de topologias.
- Executar testes de sobrecusto com balanceadores em escalas maiores, avaliando o sobrecusto de outras funcionalidades da Net Topo e como essa se comporta em outros ambientes.
- Buscar uma inicialização que contenha informações de latência entre pontos da rede, possivelmente utilizando *netloc*.
- Aplicar um mecanismo de normalização para relativizar as distâncias na topologia, as deixando mais representativas para mecanismos de escalonamento e balanceamento.
- Produzir um mecanismo de agrupamento de regiões da rede, similar ao que foi feito para agrupamento de nós e máquinas, permitindo uma redução nos cálculos de distância e outro nível de proximidade, útil para redes no estilo *Dragonfly* que possuem partes da rede com latências distintas.
- Investigar alternativas ao uso de uma memoização como matriz e avaliar seus benefícios. Como por exemplo um mecanismo de cache um ou um uso de caminhos pré calculados.

Referências

AMDAHL, G. M. Validity of the single processor approach to achieving large scale computing capabilities. *AFIPS Conference Proceedings*, AFIPS, Atlantic City, N.J., v. 30, n. 3, p. 483–485, Summer 1967. ISSN 1098-4232.

BECKER, A.; ZHENG, G.; KALÉ, L. V. Load balancing, distributed memory. In: PADUA, D. (Ed.). *Encyclopedia of Parallel Computing*. Boston, MA: Springer US, 2011. p. 1043–1051. ISBN 978-0-387-09766-4.

BESTA, M.; HOEFLER, T. Slim fly: A cost effective low-diameter network topology. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. Piscataway, NJ, USA: IEEE Press, 2014. (SC '14), p. 348–359. ISBN 978-1-4799-5500-8. Disponível em: <https://doi.org/10.1109/SC.2014.34>.

BHATELE, A. *Automating Topology Aware Mapping for Supercomputers*. Tese (Doutorado) — Dept. of Computer Science, University of Illinois, August 2010. <http://hdl.handle.net/2142/16578>.

BHATELE, A. Topology aware task mapping. In: PADUA, D. (Ed.). *Encyclopedia of Parallel Computing*. Boston, MA: Springer US, 2011. p. 2025–2025. ISBN 978-0-387-09766-4.

BHATELE, A. et al. Analyzing network health and congestion in dragonfly-based supercomputers. In: *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. Chicago, IL, USA: IEEE, 2016. p. 93–102. ISSN 1530-2075.

BHATELE, A.; KALÉ, L. V.; KUMAR, S. Dynamic topology aware load balancing algorithms for molecular dynamics applications. In: *Proceedings of the 23rd International Conference on Supercomputing*. New York, NY, USA: ACM, 2009. (ICS '09), p. 110–116. ISBN 978-1-60558-498-0.

BROQUEDIS, F. et al. hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In: IEEE (Ed.). *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*. Pisa, Italy: [s.n.], 2010. Disponível em: <https://hal.inria.fr/inria-00429889>.

CULLER, D.; SINGH, J.; GUPTA, A. Interconnection network design. In: *Parallel Computer Architecture: A Hardware/Software Approach*. 1st. ed. Burlington, Massachusetts: Morgan Kaufmann, 1998. cap. 10, p. 675–750. ISBN 1558603433.

DUPROS, F. et al. High-performance finite-element simulations of seismic wave propagation in three-dimensional nonlinear inelastic geological media. *Parallel Computing*, v. 36, p. 308–325, 2010.

GOGLIN, B.; HURSEY, J.; SQUYRES, J. M. Netloc: Towards a comprehensive view of the hpc system topology. In: *2014 43rd International Conference on Parallel Processing Workshops*. Minneapolis, MN, USA: IEEE, 2014. p. 216–225. ISSN 0190-3918.

GUSTAFSON, J. L. Reevaluating amdahl's law. *Commun. ACM*, v. 31, p. 532–533, 1988.

HENNESSY, J. L.; PATTERSON, D. A. Introduction. In: *Computer Architecture, Sixth Edition: A Quantitative Approach*. 6th. ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2017. cap. 1, p. 2–67. ISBN 0128119055, 9780128119051.

KALE, L. V.; BHATELE, A. *Parallel Science and Engineering Applications: The Charm++ Approach*. 1st. ed. Boca Raton, FL, USA: CRC Press, Inc., 2013. ISBN 1466504129, 9781466504127.

KIM, J. et al. Technology-driven, highly-scalable dragonfly topology. *SIGARCH Comput. Archit. News*, ACM, New York, NY, USA, v. 36, n. 3, p. 77–88, jun. 2008. ISSN 0163-5964. Disponível em: <http://doi.acm.org/10.1145/1394608.1382129>.

KUCK, D. J. Parallel computing. In: PADUA, D. (Ed.). *Encyclopedia of Parallel Computing*. Boston, MA: Springer US, 2011. p. 1409–1416. ISBN 978-0-387-09766-4.

LEUNG, J.; KELLY, L.; ANDERSON, J. H. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. Boca Raton, FL, USA: CRC Press, Inc., 2004. ISBN 1584883979.

LI, S. et al. Low latency, high bisection-bandwidth networks for exascale memory systems. In: *Proceedings of the Second International Symposium on Memory Systems*. New York, NY, USA: ACM, 2016. (MEMSYS '16), p. 62–73. ISBN 978-1-4503-4305-3. Disponível em: <http://doi.acm.org/10.1145/2989081.2989130>.

LIU, H. et al. Application of butterfly clos-network in network-on-chip. In: *TheScientificWorld-Journal*. London, UK: Hindawi Publishing Corporation, 2014.

PILLA, L. L. *Topology-Aware Load Balancing for Performance Portability over Parallel High Performance Systems*. Tese (Doutorado) — Université de Grenoble, 2014.

PILLA, L. L.; MENESES, E. *Programação Paralela em Charm++*. 2015.

RODRIGUES, E. R. et al. A comparative analysis of load balancing algorithms applied to a weather forecast model. In: *2010 22nd International Symposium on Computer Architecture and High Performance Computing*. Petropolis, Brasil: IEEE, 2010. p. 71–78. ISSN 1550-6533.

SNIR, M. Distributed-memory multiprocessor. In: PADUA, D. (Ed.). *Encyclopedia of Parallel Computing*. Boston, MA: Springer US, 2011. p. 574–578. ISBN 978-0-387-09766-4.

SOLIHIN, Y. Interconnection network architecture. In: *Fundamentals of Parallel Computer Architecture: Multichip and Multicore Systems*. London, UK: Chapman and Hall, 2009. v. 1, cap. 12, p. 361–384. ISBN 098416300X.

SUN, J.; VANDIERENDONCK, H.; NIKOLOPOULOS, D. S. Accelerating graph analytics by utilising the memory locality of graph partitioning. In: *2017 46th International Conference on Parallel Processing (ICPP)*. Bristol, UK: IEEE, 2017. p. 181–190. ISBN 978-1-5386-1042-8. ISSN 2332-5690.

TANENBAUM, H. B. A. S. Multiple processor systems. In: *Modern Operating Systems*. 4th. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014. pdf 8, p. 517–593.

TESSER, R. K. et al. Improving the performance of seismic wave simulations with dynamic load balancing. In: *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*. Torino, Italy: IEEE, 2014. p. 196–203. ISSN 1066-6192.

University of Illinois. *Parallel Languages/Paradigms: Charm++*. 1996. Acessado em 08 nov. 2018. Disponível em: <http://charm.cs.illinois.edu/research/charm>.

University of Southern California. *Cereal - A C++11 library for serialization*. 2013. Acessado em 13 Maio. 2019. Disponível em: <https://uscilab.github.io/cereal/index.html>.

APÊNDICE A – Códigos da Net Topo

O código criado pode ser encontrado em https://github.com/Thaleszh/Net-Topo.

APÊNDICE B – Artigo

Net Topo: uma Abstração da Topologia de Rede para Escalonamento para Aplicações Paralelas e Distribuídas

Thales A. Z. Hübner[1]

[1]Departamento de Informática e Estatística – Universidade Federal de Santa Catarina – Floriánopolis – SC – Brasil

thales.z.h@grad.ufsc.br

Abstract. *Net Topo is a network topology abstraction that grants easy access to topology information and eases their use on load balancing. It offers proximity and distance functions between network nodes, with focus on reducing contention and communication latency.*
Net topo brought benefits in initialisation time while adding no substantial cost. By storing topology information for future use it reduced the topology start time by 97%. The performance of this abstraction was observed in different scales, showing that Net Topo scales well in the majority of its functions.

Resumo. *Este trabalho desenvolveu a Net Topo, uma abstração da topologia de rede que facilita o acesso a informações de topologia da rede e a utilização destas para distribuição de tarefas. Ela oferece funções de proximidade e distância entre nós na rede com foco em contenção e latência de comunicação.*
A abstração elaborada não só teve sobrecusto negligenciável mas apresentou benefícios na inicialização e na velocidade da estrutura. Por arquivar as informações de topologia para execuções futuras, a Net Topo permitiu uma redução de 97% no tempo de inicialização de topologia. O desempenho da abstração foi avaliado em diferentes escalas, mostrando que a Net Topo consegue apresentar boa escalabilidade na maioria das suas funções.

1. Introdução

A alocação de tarefas em um sistema geralmente ocasiona um uso compartilhado de *links* por múltiplos núcleos, que pode saturar partes da rede devido a uma carga maior de informação a ser transmitida do que os *links* suportam, resultando em contenção [Bhatele 2011]. Outro fator de topologia que impacta o desempenho da aplicação é a possibilidade da rede ter ligações com custos heterogêneos [Bhatele et al. 2016], tendo latências diferentes para *links* diferentes.

A repartição de trabalho em ambientes paralelos é um problema a ser tratado, pois aplicações como simulações sísmicas [Dupros et al. 2010, Tesser et al. 2014], dinâmica molecular [Bhatele et al. 2009] ou previsão de tempo [Rodrigues et al. 2010] possuem comportamento dinâmico, ou seja suas cargas mudam ao longo da execução da aplicação, levando um mapeamento homogêneo de tarefas a um estado desbalanceado do sistema, resultando em uma diferença de carga entre alguns PEs. Esta diferença faz com que alguns PEs não tenham trabalho para executar enquanto esperam a conclusão de tarefas nos PEs mais carregados, reduzindo o desempenho geral da aplicação. Para consertar tais desvios pode-se usar um balanceador de carga (*Load*

Balancer ou LB) dinâmico, cujo trabalho é remapear as tarefas entre os PEs, buscando um estado mais balanceado do sistema. Este rearranjo garante a utilização dos PEs de maneira mais homogênea e leva a uma redução do tempo da aplicação pela redução de tempo ocioso.

Os caminhos utilizados dentro de uma rede impactam o tempo de execução de um programa, pois caminhos mais lentos resultam em uma latência maior de comunicação e problemas como contenção. É possível que aplicações e LBs levem em conta os custos de comunicação entre os PEs e os custos de movimentação destas tarefas dentro da rede. Tal abordagem pode levar a uma redução do custo de comunicação, evitando a troca de informação e tarefas através de partes lentas da rede. Uma abstração da topologia de rede que permita fácil acesso às informações da rede e de comunicação reduziria a complexidade de criação de aplicações que lidam com contenção e latência.

Hardware Locality (*hwloc*) e *Network Locality* (*netloc*) são duas ferramentas que fornecem uma abstração da topologia para aplicações. Ambas coletam diversas informações de um sistema, realizam uma interpretação intermediária e a disponibilizam para um usuário, de maneira visual ou em um arquivo *Extensible Markup Language* (XML). O *hwloc* apanha somente a parte de topologia de máquina, enquanto o *netloc* coleta informações da rede. Nenhuma das duas visa auxiliar alocação de tarefas com funções para cálculo de proximidade ou distâncias.

Este trabalho desenvolveu a Net Topo, uma abstração de topologia com foco em auxiliar na alocação de tarefas, detalhado adiante.

2. Abstrações de topologia

Nesta seção são abordados dois trabalhos que abstraem informações de topologia para seu uso em aplicações: o *hwloc* [Broquedis et al. 2010] e o *netloc* [Goglin et al. 2014].

Hwloc é um *software* que provê uma abstração da hierarquia das arquiteturas de máquinas, dispondo uma série de informações de *cache*, núcleos, *multithreading*, *sockets* e dispositivos de entrada e saída [Broquedis et al. 2010]. Seu objetivo é facilitar o acesso de informações complexas das arquiteturas atuais, de maneira uniforme e portável para aplicações. Em questão de topologia, o *hwloc* trabalha somente em um ambiente multiprocessado, em uma única máquina, não oferecendo suporte em relação a topologia de um ambiente multicomputado ou distribuído.

Netloc é um projeto de *software* acoplado ao *hwloc* que provê uma abstração da topologia de rede. Assim como o *hwloc*, o *software* é usado para encontrar informações completas da topologia, ainda não descobertas, e disponibilizá-las para o usuário. O trabalho do *netloc* é voltado para métodos de descobrimento e representação da topologia de rede, unindo estas informações com as informações de máquina do *hwloc* [Goglin et al. 2014].

O projeto *netloc* ainda está em desenvolvimento e ainda não possui representação de distância ou latência entre áreas distintas da rede. Além disso, sua preocupação é de fornecer todas as informações que conseguir sobre a topologia utilizada de maneira dinâmica para considerar alterações na rede devido a falhas e expan-

sões [Goglin et al. 2014]. A quantidade e complexidade destas informações aumentam a ocupação de memória e o tempo de inicialização da abstração, adicionando um custo indesejado.

2.1. Charm++

Charm++ [University of Illinois 1996] é um sistema de programação paralela independente de máquina desenvolvido na *University of Illinois at Urbana-Champaign* [Kale and Bhatele 2013]. O sistema é baseado na ideia de objetos migráveis chamado de *chares* cuja execução é guiada pela troca de mensagens. O seu sistema de execução (*runtime*) possibilita a execução de programas paralelos com mecanismos avançados, como balanceamento de carga automático e *checkpointing*.

Para garantir a eficiência na troca de mensagens entre suas tarefas, a plataforma Charm++ adquire as informações de topologia da rede com o seu *runtime*. No início da execução de uma aplicação, é realizado uma troca de mensagens entre os processos em execução em busca de suas conexões. Esta informação é utilizada para inferir a topologia utilizada dentre a seguinte lista: *mesh*, *fatt-tree*, torus, anel e grafo completo.

3. Net Topo

O principal objetivo da Net Topo é auxiliar na alocação e distribuição de carga fornecendo informações de uma topologia de rede. Para que a abstração possa ser utilizada independente da topologia de um sistema, sua representação deve ser flexível e genérica, podendo comportar tipos diferentes de topologias sem perder funcionalidades.

A topologia sendo representada pela Net Topo é dividida em duas camadas: uma de máquina, representada em árvores, e uma de rede, representada em um *Compressed Sparse Columns*. Por último é armazenado resultados de cálculos de distância em uma matriz de memoização. A Figura 1 apresenta um exemplo de topologia e sua representação na Net Topo.

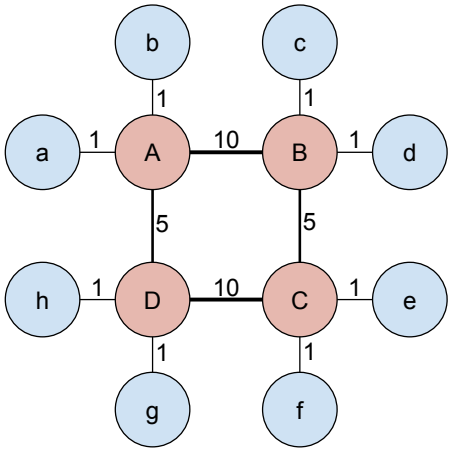
3.1. Funcionalidades

O principal uso das informações de topologia para um escalonador ou balanceador de carga é indicar a proximidade entre tarefas, visando a redução de latência e contenção. Para suprir esta informação, a abstração possui três funções de proximidades: uma relativa a estar na mesma máquina ou não, uma relativa a quantidade de *hops* e uma última em relação a distância mínima, levando em conta pesos na rede. Para reutilizar cálculos de distâncias é utilizada uma matriz de memoização para armazenar resultados.

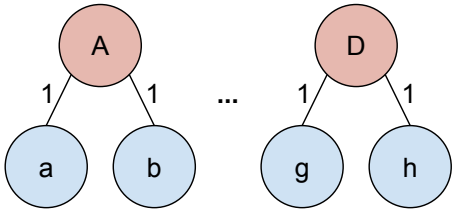
Além destas três funções de proximidade, a Net Topo é capaz de suprir uma série de informações relativas a posicionamento de máquinas e processadores e tamanho da topologia.

3.2. Serialização

Um outro objetivo da abstração é oferecer persistência de informação entre execuções no mesmo sistema, seguindo a ideia do *hwloc* de arquivar uma topologia. Esse arquivamento permite uma importação posterior das informações de topologia



(a) Topologia.



(b) Árvore.

Índice	0	2	4	6	8			
Linha	B	D	A	C	B	D	A	C
Distância	10	5	10	5	5	10	5	10

(c) CSC.

M	A	B	C	D
A	X	10		5
B	X	X	5	15
C	X	X	X	10
D	X	X	X	X

(d) Matriz.

Figura 1: Representação da topologia mostrada em (a)-(b) de acordo com as estruturas propostas.

sem passar novamente por um processo de descobrimento, reduzindo o sobrecusto de inicialização da estrutura [Broquedis et al. 2010].

Para realizar a serialização das estruturas da Net Topo, foi utilizado a biblioteca de serialização *Cereal* [University of Southern California 2013]. A biblioteca foi escolhida pela sua simplicidade, velocidade e pela compatibilidade com a linguagem *C++*. O arquivo de serialização gerado está na linguagem XML (*Extensible Markup Language*), o mesmo utilizada pelo *hwloc*, pois oferece legibilidade do que está armazenado.

3.3. Inicialização

Para ampliar e facilitar seu uso, a Net topo é uma abstração desacoplada de outras implementações, o que faz que sua inicialização seja dependente de alguma fonte de informações topológicas. As informações de rede podem ser obtidas manualmente ou através de mecanismos automáticos, como o *netloc*.

Neste trabalho foram criadas três inicializações: uma manual, que recebe uma entrada padrão; uma automática, utilizando um adaptador para o Charm++; e uma através de um arquivo XML gerado anteriormente pelo Net Topo.

4. Avaliação de sobrecusto

Para observar o sobrecusto da abstração, foi comparada a execução do balanceador NeighborLB, com o balanceador myNeighborLB, uma modificação do anterior que utiliza a Net Topo em vez do sistema topológico do Charm++, mas efetua o mesmo

algoritmo. O *benchmark* sintético *LB Test* foi utilizado para realizar uma avaliação da execução dos balanceadores de carga. Nele são criadas cargas uniformemente distribuídas entre uma valor de carga mínimo e máximo. O *LB test* cria um perfil de comunicação entre estas cargas e invoca um balanceador de carga periodicamente, ambos escolhidos pelo usuário.

Foram realizados três configurações do *LB test* para realizar os experimentos capturados neste trabalho, com os seguintes nomes: (i) *init*, (ii) *LB test M* e (iii) *LB test G*. A configuração (i) captura os tempos de inicialização enquanto as configurações (ii) e (iii) visam observar variação no desempenho do balanceador. Em ambas foi utilizada uma execução de 150 iterações com o balanceador sendo invocado a cada 40 iterações, o intervalo de carga foi de 30 a 4120ms e o padrão de comunicação utilizado foi de anel. A diferença entre as configurações (ii) e (iii) é o número de tarefas criadas.

4.1. Ambiente de Testes

A execução foi realizada em um conjunto de duas máquinas diferentes: (i) Ciclope e (ii) Centauro. A primeira máquina utiliza 4 CPUs Intel Core i7-7700 @3.60GHZ com *hypertreading* e possui 8GB@1200 MHz de memória. Possui 3 níveis de cache com tamanhos de 32 KB, 256 KB e 8 MB. Sua versão de GCC é 5.5.0. A segunda máquina possui 4 CPUs Intel Core i5-7400 @3.00GHz e 16GB@2400 MHz DIMM de memória. Possui 3 níveis de cache com tamanhos de 32 KB, 256 KB e 6 MB. GCC instalado na versão 5.4.0. Ambas as máquinas utilizam Linux Mint na versão 18.2 e Charm++ 6.9.0.

4.2. Inicialização

Na configuração de *init* foram observadas quatro inicializações: (i) a Nativa do Charm++, (ii) a do Net Topo inicializada com após a nativa e (iii) inicialização Net Topo com um arquivo XML. A inicialização (i) realiza uma série de mensagens IP entre as máquinas, encontrando e identificando cada PE de cada máquina as enquadrando para uma das topologia internas. A (ii) realiza a inicialização do Charm++, as organiza para a estrutura do Net Topo e as armazena, salvando tudo em um arquivo XML. Por último, a importação XML (iii) captura as informações armazenadas por uma inicialização anterior do Net Topo.

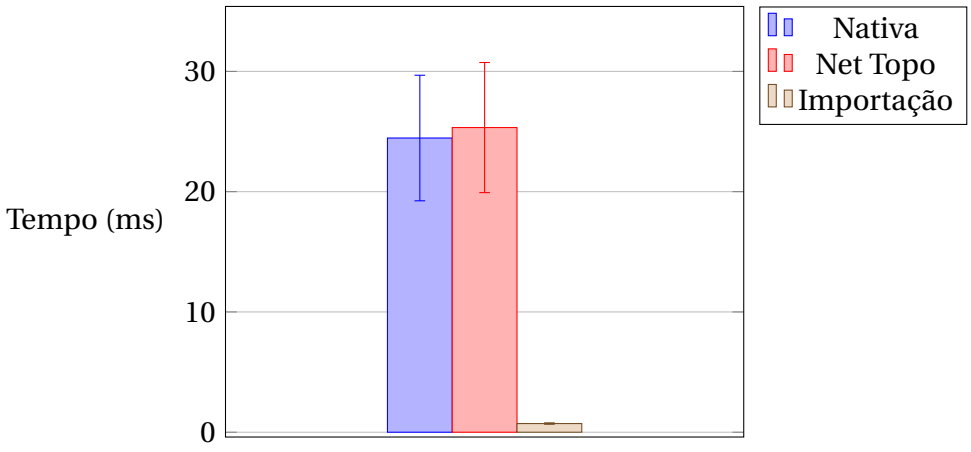


Figura 2: Inicializações testadas, seus tempos de execução e seu desvio padrão.

A Figura 2 apresenta o tempo e desvio padrão de cada uma das inicializações observadas. Quando comparada com a nativa, a adição da inicialização da Net Topo gerou em média 3.5% de sobrecusto. Quando comparada a nativa, a importação XML reduz em média o tempo de inicialização em 97.1%, agilizando o processo de inicialização do sistema. Conforme o número de PEs do sistema aumenta, é esperado que este ganho entre a importação e a inicialização cresça ainda mais, pois a implementação nativa realiza uma multitude de mensagens IPs para encontrar e definir a topologia. Por outro lado, o custo de inicialização da Net Topo deve crescer junto com o número de PEs. Não se sabe se o sobrecusto dela irá se manter, aumentar ou reduzir quando adicionada a nativa, mas se o desempenho da importação se mantiver, é provável que o sobrecusto de inicialização do Net Topo se pague na segunda execução.

4.3. Balanceamento de Carga

Os resultados de tempo dos balanceadores de carga são observados no gráfico da Figura 3. Não houve mudanças significativas no comportamento do balanceador, o que demonstra um comportamento desejado pela abstração.

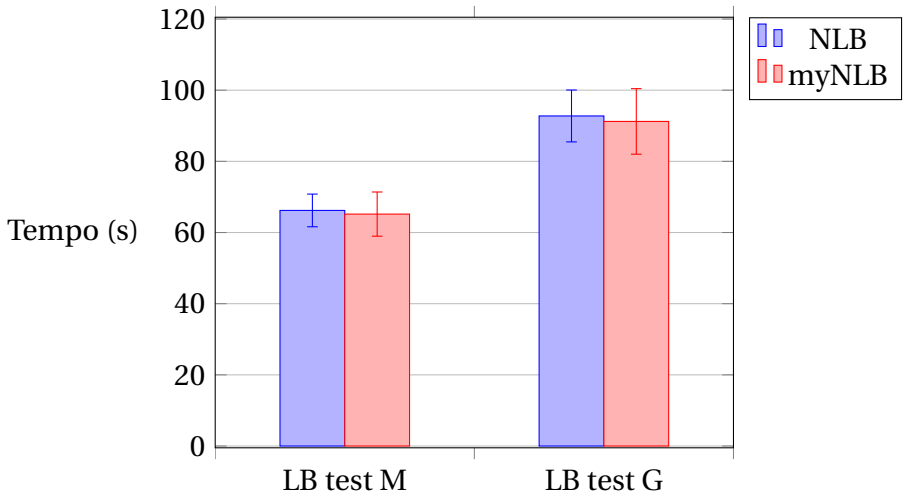


Figura 3: Comparação de tempos de balanceamento entre NeighborLB e myNeighborLB. Os intervalos em cada barra representam o desvio padrão.

5. Avaliação de escala

Foram criados 4 casos de representações de topologias para a execução dos testes: uma *fat-tree* com 7 níveis, nomeada de *tree M*; uma *fat-tree* de 10 níveis, representada como *tree G*; uma *mesh2D* de tamanho 16x16, representada por *mesh2D*; e uma *mesh3D* simétrica de dimensão 8, nomeada *mesh3D*. Todas as topologias tiveram uma versão da abstração com memoização e outra sem. O número de máquinas e ligações de cada um dos casos de teste estão apresentados na Tabela 1. A coluna tamanho do arquivo XML e XML- representa o espaço em disco do arquivo XML gerado pela Net Topo com e sem memoização, respectivamente. É possível observar que este tamanho não escala bem devido ao dispositivo de memoização, por ocupar espaço com complexidade $O(m^2)$ onde m é o número de maquinas.

Caso	Número de máquinas	Número de ligações	XML/M	XML
tree M	127	126	0,32	0,05
mesh2D	256	538	1,28	0,12
mesh3D	512	1.344	5,06	0,31
tree G	1.023	1.022	20,06	0,42

Tabela 1: Tamanho das abstrações criadas em número de máquinas, em número de ligações e em ocupação do arquivo XML em MB com e sem memoização, representado por XML/M e XML, respectivamente.

A execução dos testes foi realizada na máquina Centauro, apresentada na Seção 4.1, utilizando um único núcleo e medindo o tempo através da biblioteca *crono* de *C++*.

5.1. Inicialização e Serialização

A Figura 4 apresenta o crescimento do tempo médio de inicialização, arquivamento e importação XML de cada um dos casos com o crescimento de número de máquinas. As topologias *tree M*, *mesh2D*, *mesh3D* e *tree G* são representadas por 2^7, 2^8, 2^9 e 2^{10} máquinas, respectivamente. É possível observar que, quando sem memoização, os tempos de arquivar e importar crescem linearmente ou sublinearmente com a quantidade de máquinas. Por outro lado é fácil observar que o mecanismo de memoização em forma de matriz não escala bem com o tamanho da topologia devido a sua natureza quadrática. Isso se deve pelo fator da matriz de memoização representar metade de todas a ligações possíveis, que são mais de 524 mil no caso *tree G*.

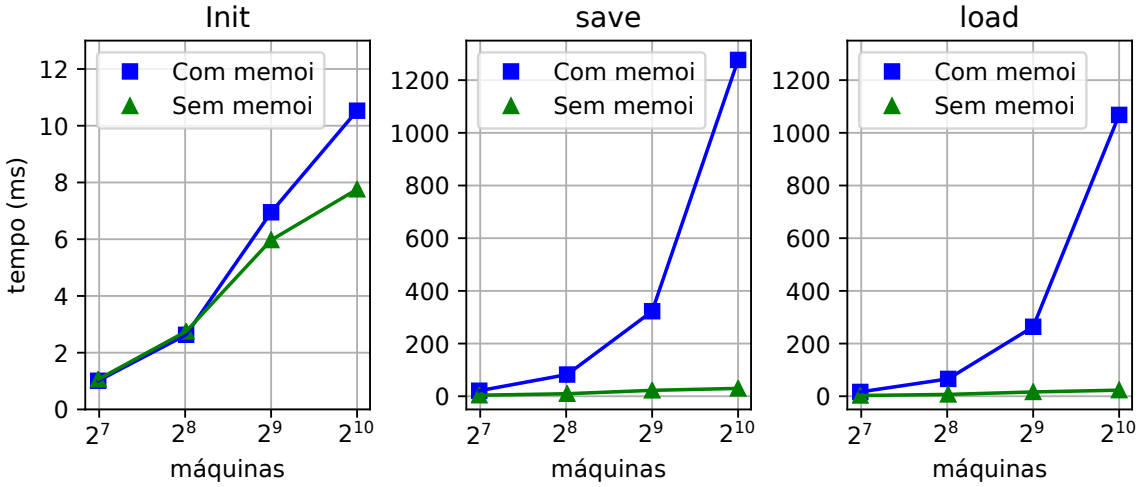


Figura 4: Crescimento dos tempo médio de inicialização (init), arquivamento (save) e importação XML (load) de cada um dos casos de teste. Os quadrados em azul são os casos com memoização e os triângulos em verde os casos sem memoização.

5.2. Distância

O cálculo de distância avalia a latência entre duas máquinas da topologia. Para avaliar a latência deste cálculo, foi agrupado o tempo de todos os cálculos de distân-

cia entre uma máquina e todas as máquinas a até dois *hops* de distância dela. Este conjunto de distâncias calculadas foi denominado como "distância II".

O cálculo da distância II utiliza o mecanismo de memoização, e foram avaliados três cenários relativos a memoização: (i) inicialmente completa, (ii) inicialmente vazia e (iii) sem nenhum tipo de memoização. A primeira contêm todas as informações de distância calculadas previamente. A segunda inicia cada cálculo com a memoização inicialmente vazia, expandindo durante a execução de uma distância II. O terceiro caso não utiliza nenhuma memoização e é utilizada para comparação.

A Tabela 2 apresenta os tempos médios de execução do preenchimento da matriz de memoização e dos três cenários de distância II. A primeira coluna apresenta o tempo de preenchimento, que é o tempo o mecanismo de memoização através de cálculos de distância entre todos os nós. A primeira linha de cada caso de execução indica a média dos tempos e a segunda linha o desvio padrão.

Caso		**Preenchimento** (ms)	**Completa** (μs)	**Vazia** (μs)	**Sem memoização** (μs)
tree M	$\overline{x}$	75,70	1,30	11,32	50,56
	σ	0,98	0,04	4,68	13,15
mesh2D	$\overline{x}$	863,91	2,69	75,72	210,42
	σ	5,70	0,04	24,04	41,77
mesh3D	$\overline{x}$	4.880,94	4,96	214,90	873,95
	σ	19,03	0,35	21,12	77,39
tree G	$\overline{x}$	6.215,44	1,40	16,63	39,49
	σ	24,24	0,06	5,74	9,63

Tabela 2: O tempo médio ($\overline{x}$) e o desvio padrão (σ) de preenchimento completo da matriz de memoização (primeira coluna), e tempo do cálculo de distância II de uma máquina com memoização completa (coluna II), inicialmente vazia (coluna III) ou inexistente (última coluna).

Todos os cálculos de distância têm um desvio padrão aumentado devido a variação do grau de cada máquina, sendo de 1 a 3 em uma *fat-tree*, 2 a 4 numa *mesh2D* e 3 a 6 em uma *mesh3D*. O desvio padrão maior no caso de memoização inicialmente vazia se deve ao fator dos cálculos oscilarem entre encontrar um valor armazenado e encontrar um valor novo.

Como o número de distâncias possíveis cresce quadraticamente com o número de máquinas, o tempo de preenchimento também cresce quadraticamente. Este crescimento é observável na Tabela 2. Para que este tempo de preenchimento acelere a execução, é necessário que seu tempo seja pago por um número grande de cálculos de distância. No caso *tree M*, por exemplo, são necessárias 6180 execuções, enquanto para *tree G* são necessárias 125845 execuções.

Quando comparado o tempo de preenchimento da *mesh3D* com da *tree G*, este não cresceu tanto quando comparado com o crescimento entre *mesh2D* e *mesh3D*, por realizar um cálculo de distância com um grau médio menor. Portanto o tempo de preenchimento da memoização cresce tanto com o número de máquinas na topologia quanto com o tempo de um cálculo de distância. Com isso, a quantidade de execuções

de distância para que o preenchimento valha a pela varia para cada topologia.

É possível observar um aumento no tempo conforme o grau da topologia aumenta, assim como uma melhoria entre a completa para a inicialmente vazia e outra melhoria entre a inicialmente vazia e a sem memoização. A melhoria no caso da inicialmente vazia se deve ao algoritmo de distância armazenar a distância de todos os visitados, conseguindo encontrar a distância de uma parte da vizinhança nas primeiras execuções e utilizá-la nas seguintes.

6. Conclusão

Neste trabalho foi desenvolvida a Net Topo, uma abstração de topologia que visa recolher e dispor informações da topologia de rede para uso em escalonamento e balanceamento de aplicações paralelas e distribuídas. Dentre diversas opções de estruturas de dados e representação de topologia, foi utilizado um CSC, uma matriz de distâncias opcional e uma lista de árvores. A abstração criada apresenta diversas informações de topologia e fornece funções de proximidade e distâncias, visando auxiliar o processo de distribuição de carga. A Net Topo também possui diversos mecanismos para sua inicialização, incluindo implementações manuais e importação de um aquivo de topologia em XML.

Foi criado um adaptador para receber informações de inicialização e fazer uma ponte entre a abstração e o *runtime* do Charm++, permitindo utilizar a Net Topo junto ao módulo de balanceamento de carga presente na plataforma. Para comprovar a integração e avaliar o sobrecusto da abstração, foi comparado um balanceador de carga com sua versão modificado para utilizar a Net Topo. Apesar das execuções serem realizadas em baixa escala e em um ambiente pequeno, elas permitiram observar um sobrecusto de inicialização altamente compensado pelo arquivamento da topologia e pouca ou nenhuma mudança negativa nos tempos de execução do balanceador.

Por fim, foi avaliado o impacto da memoização e observado a escalabilidade da abstração. O dispositivo de memoização através de uma matriz completa não se mostrou escalável, gerando sobrecustos muito altos conforme o tamanho da topologia cresce. A função de proximidade se apresentou como constante e o restante das funções testadas demonstraram crescimento linear ou sublinear com o aumento no número de máquinas.

7. Trabalhos Futuros

A execução deste trabalho criou uma série de oportunidades de trabalhos futuros. Segue uma lista de extensões, melhorias e avaliações cogitadas:

- Realizar uma distribuição das estruturas de topologia de modo que não seja puramente replicada em cada nó e reduza o espaço utilizado em cada máquina, mas ainda mantenha a possibilidade de uma travessia de grafo para cálculo de *hops* ou latência.
- Modificar as estruturas da abstração para serem dinâmicas, possibilitando modificações e atualizações em uma topologia prévia, sem ter de reinicializá-la.
- Implementar uma inicialização utilizando informações do *hwloc* para a topologia de máquina e o Charm++ para a topologia de rede, retirando as imprecisões

relacionadas a máquinas do Charm++ enquanto se mantém a automatização do processo.
- Aplicar um mecanismo de normalização para relativizar as distâncias na topologia, as deixando mais representativas para mecanismos de escalonamento e balanceamento.
- Produzir um mecanismo de agrupamento de regiões da rede, permitindo uma redução nos cálculos de distância e outro nível de proximidade, útil para redes no estilo *Dragonfly* que possuem partes da rede com latências distintas.
- Investigar alternativas ao uso de uma memoização como matriz e avaliar seus benefícios. Como por exemplo um mecanismo de cache um ou um uso de caminhos pré calculados.

Referências

Bhatele, A. (2011). Topology aware task mapping. In Padua, D., editor, *Encyclopedia of Parallel Computing*, pages 2025–2025. Springer US, Boston, MA.

Bhatele, A., Jain, N., Livnat, Y., Pascucci, V., and Bremer, P. T. (2016). Analyzing network health and congestion in dragonfly-based supercomputers. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 93–102, Chicago, IL, USA. IEEE.

Bhatele, A., Kalé, L. V., and Kumar, S. (2009). Dynamic topology aware load balancing algorithms for molecular dynamics applications. In *Proceedings of the 23rd International Conference on Supercomputing*, ICS '09, pages 110–116, New York, NY, USA. ACM.

Broquedis, F., Clet-Ortega, J., Moreaud, S., Furmento, N., Goglin, B., Mercier, G., Thibault, S., and Namyst, R. (2010). hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications. In IEEE, editor, *PDP 2010 - The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing*, Pisa, Italy.

Dupros, F., Martin, F. D., Foerster, E., Komatitsch, D., and Roman, J. (2010). High-performance finite-element simulations of seismic wave propagation in three-dimensional nonlinear inelastic geological media. *Parallel Computing*, 36:308–325.

Goglin, B., Hursey, J., and Squyres, J. M. (2014). Netloc: Towards a comprehensive view of the hpc system topology. In *2014 43rd International Conference on Parallel Processing Workshops*, pages 216–225, Minneapolis, MN, USA. IEEE.

Kale, L. V. and Bhatele, A. (2013). *Parallel Science and Engineering Applications: The Charm++ Approach*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition.

Rodrigues, E. R., Navaux, P. O. A., Panetta, J., Fazenda, A., Mendes, C. L., and Kale, L. V. (2010). A comparative analysis of load balancing algorithms applied to a weather forecast model. In *2010 22nd International Symposium on Computer Architecture and High Performance Computing*, pages 71–78, Petropolis, Brasil. IEEE.

Tesser, R. K., Pilla, L. L., Dupros, F., Navaux, P. O. A., Méhaut, J. F., and Mendes, C. (2014). Improving the performance of seismic wave simulations with dynamic load balancing. In *2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, pages 196–203, Torino, Italy. IEEE.

University of Illinois (1996). Parallel languages/paradigms: Charm++. Acessado em 08 nov. 2018.

University of Southern California (2013). Cereal - a c++11 library for serialization. Acessado em 13 Maio. 2019.