**UNIVERSIDADE FEDERAL DE SANTA CATARINA**
**DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA**

Igor da Silva Solecki

# REPRESENTAÇÃO OTIMIZADA DE TOPOLOGIAS DE MÁQUINA

Florianópolis

2016

Igor da Silva Solecki

# REPRESENTAÇÃO OTIMIZADA DE TOPOLOGIAS DE MÁQUINA

Trabalho de Conclusão de Curso submetido ao Curso de Bacharelado em Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador: Prof. Dr. Laércio Lima Pilla
Universidade Federal de Santa Catarina

Florianópolis

2016

Igor da Silva Solecki

**REPRESENTAÇÃO OTIMIZADA DE TOPOLOGIAS DE MÁQUINA**

Este Trabalho de Conclusão de Curso foi julgado aprovado para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovado em sua forma final pelo Curso de Bacharelado em Ciências da Computação.

Florianópolis, 08 de Dezembro 2016.

---

Prof. Dr. Mário Antônio Ribeiro Dantas
Coordenador
Universidade Federal de Santa Catarina

**Banca Examinadora:**

---

Prof. Dr. Laércio Lima Pilla
Universidade Federal de Santa Catarina
Orientador

---

Prof. Dr. Luiz Cláudio Villar dos Santos
Universidade Federal de Santa Catarina

---

Prof. Dr. Márcio Bastos Castro
Universidade Federal de Santa Catarina

# LISTA DE FIGURAS

# LISTA DE SIGLAS E ABREVIATURAS

**SUMÁRIO**

# 1 INTRODUÇÃO

Atualmente, arquiteturas de computadores são construídas de forma hierárquica quanto à memória. Essa hierarquia diz respeito à passagem de dados entre os diferentes níveis, ou seja, quais caminhos existem para que dados sejam comunicados entre pontos dessa hierarquia. Níveis de memória mais altos possuem maior capacidade de armazenamento, porém seu tempo de acesso é maior. Parte dessa hierarquia é composta por um ou mais níveis de cache, memórias com capacidade reduzida, mas maior velocidade, permitindo que, em um dado momento, um conjunto de dados qualquer possa ser acessado mais rapidamente. No nível mais baixo dessas hierarquias estão as unidades de processamento, acessando e operando sobre os dados em memória. Quanto mais próximo for o nível de memória em que esse dados estiverem, menor o tempo de acesso. Quando essas unidades precisam se comunicar entre si, elas fazem uso da hierarquia de memória.

A hierarquia pode ser organizada de várias formas, podendo se tornar complexa e de grande profundidade. Uma possibilidade na organização é o compartilhamento de alguns níveis de cache, ou seja, duas unidades de processamento ou mais estarão debaixo de uma mesma cache na hierarquia. Isso permite, por exemplo, realizar comunicações com eficiência, pois o tempo entre algum dado ser atualizado e o novo valor ser visto é determinado pelo tempo de acesso à cache compartilhada. Uma grande variedade de organizações pode ser encontrada ao se considerar arquiteturas como multicore, em que várias unidades de processamento, que recebem o nome de núcleos (cores), estão físicamente agrupadas de algum modo, podendo haver compartilhamento de cache, ou NUMA (Non-Uniform Memory Access), em que mais níveis são acrescentados à hierarquia, os quais compõe a rede de interconexão, a qual, por si só, também pode ser organizada de várias formas distintas. Essa organização compreendendo hierarquia de memória e unidades de processamento, na sua totalidade, define uma topologia de máquina.

A necessidade de plataformas para rodar aplicações de alto desempenho tem dado origem às diversas arquiteturas paralelas modernas existentes. Suas topologias são as mais variadas, visando atender às necessidades de várias classes de aplicações com características e comportamentos distintos. Diante da crescente complexidade das topologias dessas máquinas, a sua organização e as demais características dos elementos que compõe a hierarquia de memória são aspectos de muita relevância para o desempenho de aplicações.

Certas combinações de fatores da aplicação e da arquitetura podem resultar na melhoria ou na degradação do desempenho. Tais fatores podem ser, por exemplo, a quantidade de dados manipulados e o tamanho das caches, que podem comportar ou não todos os dados simultaneamente, ou os padrões de troca de mensagens entre tarefas e a localização delas, além dos meios existentes para realizar essas comunicações, que podem resultar em maior ou menor eficiência.

Ainda, em arquiteturas NUMA, nas quais diferentes regiões da memória possuem diferentes tempos de acesso, é importante que haja proximidade entre os dados acessados por uma thread e o núcleo em que ela reside. Portanto, é essencial o conhecimento da topologia da máquina, que possibilita o devido ajuste das aplicações a ela, de modo a aproveitar ao máximo os recursos disponíveis.

Disso vem a necessidade de haver alguma representação da topologia para fornecer as informações necessárias sobre ela, seja diretamente às aplicações ou a outras partes do sistema, que usarão tais informações para realizar otimizações estática ou dinamicamente. Como exemplo de uso estático, pode-se citar compilação de algoritmos com conhecimento da hierarquia [Sequoia], ou posicionamento de processos MPI [hwloc-2010]; e, quanto ao uso dinâmico, posicionamento de threads e dados OpenMP [FGOMP].

No entanto, a disponibilização de tais informações gera custos adicionais, além de ter outras implicações relacionadas ao tamanho das estruturas de dados que podem afetar o desempenho. Assim, é necessário que haja um compromisso entre o tempo de acesso e o espaço ocupado pela representação utilizada. Tempos de acesso muito grandes podem acabar anulando os ganhos das otimizações. Já se as estruturas de dados forem muito espaçosas, pode ser que não possuam boa localidade espacial, dependendo dos padrões de referência aos dados em acessos consecutivos. Isso pode resultar em perda de desempenho ocasionada por faltas de cache, tanto no acesso às informações da topologia, que estarão espalhadas em diversas posições da cache, podendo ser substituídas com maior probabilidade, quanto no acesso pelas aplicações aos seus próprios dados. Entretanto, é possível que a adição de algumas informações facilitem certas consultas sobre a topologia sem causar tais prejuízos, que é o desejado.

## 1.1 MOTIVAÇÃO

Os exemplos de usos estáticos e dinâmicos dados acima, além de vários outros existentes, com o uso de benchmarks, servem como justificativa para a realização de esforços para desenvolver reprentações com as características citadas, isto é, bom tempo de acesso e uso eficiente da memória.

Visto que, como dito anteriormente, os níveis de cache inferiores são mais rápidos, o ideal é que os dados estejam sempre nos níveis os mais próximos possíveis, de modo que seu uso nas computações seja mais eficiente. Diante disso, compilação com conhecimento da hierarquia [citar de novo?] se vale do fato de que frequentemente problemas podem ser divididos em problemas menores de tamanho variável, e ajustar esses tamanhos à capacidade das caches torna o uso delas mais efetivo, pois todos os dados usados nessas partes menores da computação caberão nelas. Ainda, quando é possível haver vários níveis de subdivisão do problema, formando também uma espécie de hierarquia de subdivisões, os tamanhos das partes em diferentes níveis podem ser ajustados aos níveis de cache consecutivos. Isso pode ser visualizado com facilidade no exemplo de multiplicação de grandes matrizes presente no artigo referenciado [ou aqui].

A velocidade de níveis de cache mais próximos também beneficia a comunicação. Portanto, em conjunto com dados sobre os padrões de comunicação entre processos, as informações sobre compartilhamento de caches podem ser usadas para definir um posicionamento de processos MPI que favoreça as comunicações [hwloc-2010]. Outra otimização possível é o uso de metódos específicos do hardware para realizar comunicações dentro de um nodo.

No contexto de arquiteturas NUMA, para diminuir o tempo de acesso a memórias remotas, há a possibilidade de mover os dados para outro nodo ou as threads para outros núcleos. Seguindo o princípio de realizar um combinação dessas opções com base nos níveis da topologia, o posicionamento dinâmico de threads e dados desenvolvido no ForestGOMP [hwloc-2014], uma extensão de uma implementação de OpenMP, se mostrou efetivo. Um cenário apresentado é a existência de vários conjuntos de threads e dados com grande afinidade, em que a migração de uma thread para outro núcleo só ocorreria se houvesse um nível de cache compartilhado, de modo a manter a thread próxima dos seus dados, enquanto em outros casos poderia haver a migração de todas as threads e dados relacionados.

Esses exemplos ilustram como informações sobre a hierarquia po-

dem efetivamente ser usadas para melhorar o desempenho de aplicações que seguem modelos ou estratégias em uso real, ou seja, os benchmarks utilizados possuem características encontradas na solução de problemas reais. Isso diz respeito a, por exemplo, padrões de comunicação ou distribuição de carga, que podem apresentar irregularidades e outras características presentes em aplicações científicas de diversas áreas.

## 1.2 OBJETIVOS

Este trabalho tem como objetivo o desenvolvimento de uma representação de topologias de máquina, compreendendo as estruturas de dados utilizadas e os métodos de acesso, que mantenha o compromisso necessário, conforme apresentado anteriormente, entre tempo de acesso e espaço ocupado na memória pelas estruturas de dados.

### 1.2.1 Objetivos Específicos

Os objetivos específicos são:

- Analisar fatores relevantes para a eficiência na representação de topologias
- Desenvolver representações (estruturas de dados e métodos de acesso)
- Testar as representações desenvolvidas, por meio de experimentos em diferentes máquinas, observando o uso da memória e o tempo de execução
- Disponibilizar uma nova ferramenta para a representação de topologias de máquina

## 1.3 METODOLOGIA

- Estudar organização de computadores com foco na hierarquia de memória
- Estudar como topologias de máquina são representadas em trabalhos e ferramentas do estado da arte
- Entender o protótipo utilizado no ECL (Embedded Computing Lab) até o momento

- Implementar novos métodos de armazenamento e acesso às informações
- Testar os novos métodos e estruturas de dados utilizando máquinas com topologias diferentes e avaliar os resultados

## 1.4 ORGANIZAÇÃO DO TRABALHO

As seções restantes estão organizadas da seguinte forma: No capítulo 2, serão apresentados alguns conceitos fundamentais relevantes para o trabalho. No capítulo 3, será fornecida uma visão geral do estado da arte em representação de topologias de máquina. No capítulo 4 constam o progresso atual do trabalho e o planejamento do seu prosseguimento.

# 2 FUNDAMENTAÇÃO TEÓRICA

Uma parte muito importante das hierarquias a ser considerada em questões de desempenho são as memórias cache.

## 2.1 MEMÓRIAS CACHE

Caches são memórias com o próposito de diminuir o tempo médio de acesso a outros níveis de memória. Mais especificamente, é comum haver dois ou três níveis de cache entre as unidades de processamento e a memória principal. Elas são construídas com tecnologias que as tornam mais rápidas que outros níveis acima. Porém, essa velocidade vem em troca de custo mais elevado. Por isso, elas têm espaço de armazenamento menor, além de que memórias maiores podem ter sua velocidade de acesso diminuída, o que também motiva a existência de vários níveis de cache.

O princípio de sua funcionalidade é manter à disposição dos programas, de forma rápida, aqueles dados dos quais eles precisam ou que estão usando no momento. Esses dados são disponibilizados conforme a capacidade de armazenamento da cache. Esta comumente é menor que o conjunto de todos os dados sobre os quais o programa opera, resultando na necessidade de remover alguns dados para acomodar outros.

### 2.1.1 Localidade Espacial

Quando um programa referencia determinada posição da memória, é comum que logo em seguida os dados nas posições de memória adjcentes sejam necessários também. Isso é a chamada localidade espacial. As caches levam isso em conta para beneficiar as aplicações, trazendo dos níveis de memória acima não só o dado requisitado, mas também os dados que o rodeiam, constituindo um bloco. Assim, do ponto de vista da cache, a memória é uma sequência de blocos que, quando necessários, são carregados em algum espaço disponível, ou substituem um bloco carregado previamente se não houver espaços disponíveis.

Analisar as caches ajuda a entender porque um esquema "esperto", com estruturas muito espaçosas para reduzir a quantidade de operações e acessos, poderia não funcionar bem. No pior caso, aces-

sos consecutivos poderiam ser todos a blocos diferentes, ocasionando o custo de trazer cada um para a cache e resultando na poluição da cache das aplicações, ou seja, diversos blocos com dados das aplicações seriam substituídos, tornando maior o tempo para acessá-los na próxima vez.

# 3 ESTADO DA ARTE

Hardware Locality (abreviado como hwloc) [citação] é um pacote de software amplamente utilizado para a representação de topologias de máquina com grande portabilidade. Ele contém ferramentas de linha de comando, além de permitir que aplicações acessem as informações sobre a topologia por meio de uma API na linguagem C.

## 3.1 OBJETOS

Objetos são uma abstração usada para representar todos os elementos presentes na topologia, tanto memória quanto núcleos. A hierarquia de memória é modelada como uma árvore de objetos, onde cada nível (conjunto de objetos com mesma profundidade) contém objetos de apenas um tipo. A topologia por completo é representada de forma mais detalhada por meio de várias ligações (ponteiros) entre objetos, conforme as suas relações na hierarquia. Essas relações são definidas como:

- Pai (parent): nodo pai na estrutura de árvore
- Filhos (children): nodos filhos na estrutura de árvore
- Irmãos (siblings): nodos com o mesmo pai
- Primos (cousins): Objetos numa mesma profundidade (e, portanto, de mesmo tipo); todos os objetos que compõe um determinado nível (irmãos ou não) são primos

A Figura X apresenta um exemplo dessas relações em uma dada topologia. Ela ilustra, ainda, no ramo mais a direita, o tratamento de assimetrias utilizado pelo hwloc: objetos de um mesmo tipo são emparelhados de modo a compor um mesmo nível. Portanto, objetos irmãos não estarão necessariamente no mesmo nível, e o conceito de profundidade não é exatamente o usado no contexto de estruturas de árvore, sendo possível irmãos terem profundidades diferentes.

Essas relações acrescentadas à estrutura de árvore facilitam a navegação entre objetos, em troca do espaço adicional ocupado por cada ponteiro nos objetos. Por exemplo, como indicado na figura, a partir de qualquer objeto, é possível navegar para o próximo primo (pelo ponteiro next_cousin), ou para o próximo irmão (ponteiro next_sibling),

se existirem. Além disso, todos os objetos são armazenados numa estrutura de array de arrays de objetos, semelhante a uma matriz, mas com arrays internos de tamanho variável. Cada um dos arrays internos contém os objetos de um nível, e eles são organizados no array externo na ordem dos níveis, de modo que é possível acessar o enésimo objeto de um dado nível diretamente usando essa estrutura.

## 3.2 CONJUNTOS DE CPUS

Cada objeto pode ter um cpuset (conjunto de CPUs), que é um mapeamento dos núcleos existentes para bits (bitmap), usado para determinar quais núcleos estão sob o objeto na hierarquia, implementados como uma sequência de variáveis de 32 bits, tantas quantas forem necessárias. A implementação dos bitmaps poderia ser otimizada para diminuir a quantidade de variáveis utilizadas em casos em que existam grandes quantidades de núcleos. Algo nesse sentido é citado no respectivo arquivo fonte em um comentário sobre otimizações que poderiam ser realizadas.

# 4 ANDAMENTO DO TRABALHO

Até o momento, foi feita uma análise de diversas funções do hwloc e suas complexidades. Para alguns pontos em específico, já estão sendo consideradas ideias que poderiam torná-los mais eficientes.

Quanto aos próximos passos, pretende-se dar atenção a funções de mais relevância: Foram identificadas as chamadas mais importantes a funções do hwloc no HieSchella [ref] e, sendo viável, uma análise semelhante poderá ser feita em outros projetos com código aberto que usam o hwloc. A princípio, o foco será posto em hierarquias simétricas, podendo haver eventuais otimizações direcionadas a esse tipo de hierarquias, tratando-se assimetria se possível posteriormente. Além disso, pretende-se levar em consideração a representação de hierarquias com grandes quantidades de núcleos, com as quais se tornarão mais visíveis as diferenças nos resultados de testes com implementações mais ou menos eficientes.