PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS

Programa de Pós-Graduação em Informática

Daniel de Araújo Santos Carmo

Avaliação de Arquiteturas Many-core e Aplicações Paralelas Utilizando Análise Formal de Conceitos

Belo Horizonte

2018

Daniel de Araújo Santos Carmo

Avaliação de Arquiteturas Many-core e Aplicações Paralelas Utilizando Análise Formal de Conceitos

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Informática.

Orientador: Prof. Dr. Henrique Cota de Freitas

Belo Horizonte

2018

FICHA CATALOGRÁFICA
Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais

Carmo, Daniel de Araújo Santos
C287a Avaliação de arquiteturas many-core e aplicações paralelas utilizando análise formal de conceitos / Daniel de Araújo Santos Carmo. Belo Horizonte, 2018.
83 f.: il.

Orientador: Henrique Cota de Freitas
Dissertação (Mestrado) – Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática

1. Arquitetura de computador. 2. Sistemas programáveis em chip. 3. Processamento paralelo (Computadores). 4. Linguagens de programação visual (Computação). 5. Conceitos - Análise. I. Freitas, Henrique Cota de. II. Pontifícia Universidade Católica de Minas Gerais. Programa de Pós-Graduação em Informática. III. Título.

CDU: 681.3-11
Ficha catalográfica elaborada por Fernanda Paim Brito– CRB 6/2999

Daniel de Araújo Santos Carmo

AVALIAÇÃO DE ARQUITETURAS MANY-CORE E APLICAÇÕES PARALELAS UTILIZANDO ANÁLISE FORMAL DE CONCEITO

Dissertação apresentada ao Programa de Pós-Graduação em Informática da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Mestre em Informática.

Prof. Dr. Henrique Cota de Freitas
(Orientador) – PUC Minas

Prof. Dr. Humberto Torres Marques Neto –
PUC Minas

Prof. Dr. Luis Enrique Zárate Gálvez –
PUC Minas

Prof. Dr. Renato Antônio Celso Ferreira –
UFMG

Belo Horizonte, 28 de fevereiro de 2018.

Dedico este trabalho a todos os que vagam sem rumo. As vezes é difícil ter um propósito.

AGRADECIMENTOS

Antes de tudo, agradeço a Deus por todas as oportunidades que ele me deu na vida. Ensinou-me que em tudo é preciso fazer a minha parte, mas com a benção Dele, tudo se torna mais fácil de ser executado.

Ao meu professor, orientador e exemplo Henrique Cota de Freitas por toda a paciência demonstrada ao longo do projeto e por sempre incentivar seus alunos a não apenas fazerem o que foi pedido mas darem o seu melhor, mostrando que oportunidades surgem para aqueles que estão preparados para ela.

Aos meus amigos da pós-graduação Matheus Alcântara Souza, Pedro Henrique de Mello Morado Penna e a todos os outros por me ajudarem a desenvolver este trabalho com suas experiências e conselhos.

À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), pela bolsa de estudos concedida, à Fundação de Amparo à Pesquisa do Estado de Minas Gerais (FAPEMIG) e ao Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) pelos auxílios e recursos computacionais utilizados.

Ao professor Jean-François Méhaut pelo auxílio na missão científica realizada em agosto de 2016 no contexto do projeto ExaSE.

"Saiba morrer o que viver não soube"

Bocage

RESUMO

A evolução das arquiteturas *many-core* em *chip* é uma tarefa complexa uma vez que o número de elementos presentes é relativamente maior do que em arquiteturas *single-core*. A avaliação convencional dessas arquiteturas é feita através da análise pura dos dados, tendo como base o conhecimento e a experiência de seu avaliador.

Assim, o processo de análise pode ficar mais lento a medida que aumenta o volume de dados para avaliar. Portanto, este trabalho apresenta uma avaliação de arquiteturas *many-core* e aplicações paralelas com utilização de Análise Formal de Conceitos. A fase de análise destes sistemas pode ser um processo lento aquando há um grande número de características para analisar. Assim, buscou-se utilizar os conceitos da AFC como uma técnica complementar com o intuito de possibilitar um maior controle das variáveis que podem impactar no desempenho de arquiteturas e aplicações paralelas. Com a AFC, é possível entender melhor a correlação entre os dados. Sua análise evidencia o quanto as características do sistema o afeta, tornando o processo de análise mais veloz.

A metodologia adotada faz uso de um simulador de sistemas completos chamado Gem5 e de um *benchmark* de aplicações paralelas chamado CAP Bench. Foram propostas dez arquiteturas many-core com *redes-em-chip*, para simulação de sete aplicações paralelas. Dentre os resultados principais é possível destacar que as arquiteturas organizadas pela topologia *cluster* apresentaram o melhor desempenho geral. Entre as aplicações paralelas, a aplicação TSP (*Traveling Salesman Problem*) possui comportamentos que ficam mais claros com o uso de AFC. Por exemplo, arquiteturas que possuam mais recursos consomem mais potência, obviamente. Porém, com a AFC, foi possível notar que as arquiteturas possuem mais recursos combinados com um uma taxa de *cache miss* alta e um baixo acesso a memória aumentam o consumo de potência. Portanto, os resultados mostraram que a AFC evidencia regras e comportamentos sobre as aplicações/arquiteturas que não ficam tão claras na avaliação convencional, uma vez que a análise dos resultados é individualizada, o que pode dificultar a correlação entre os resultados. Todavia, a AFC se torna mais importante a medida que o número de variáveis envolvidas fica maior. Nesse sentido, há uma melhor qualidade dos resultados gerados pela AFC, evidenciando regras e comportamentos que podem não ficar tão claros em uma tentativa de correlacionar pelo método convencional.

Palavras-chave: Arquiteturas de Processadores *Many-Core*, Redes-em-Chip, Aplicações Paralelas, Análise Formal de Conceitos, Simulação de Sistemas Completos.

ABSTRACT

The evolution of many-core architectures is a hard task. Comparing with single-core architectures, the great number of elements present in many-core improves the complexity of their analysis. The conventional evaluation is done through the pure analysis of data and the experience and knowledge of the appraiser.

Because of that, the analysis proccess is slow while the data size for evaluation scales. In view of this fact, this work shows a many-core architectures evaluation with parallel aplication using FCA (Formal Concept Analysis). Thus, it was tried to use the FCA generated concepts as a complementary technique to provides a higher understanding for the variables that can impactate on the architectures and parallel applications performances.

The adopted methodology uses the full-system simulator GEM5 and a parallel application benchmark called CAP bench. It was proposed ten many-core architectures with network-on-chip and seven parallel applications. Among the main results its possible highlight the cluster-based architectures, which achieves the best overall results. Among the parallel applications, traveling salesman problem has more clear behaviors with FCA analysis, such as architectures with more resources will achieve a higher power consuption only with a low memory consumption in the most cases, showing that the adding of resources combined with a high cache miss ratio and a low memory access will increase the architecture energy consumption. Therefore, the results showed that FCA evidences rules and behaviors about architectures and applications are not so clear for conventional evaluation once the result analysis is individualized, what could embarrass the correlation between the results. However, the FCA evaluation become more important as the number of analyzed variable increases. In this way, there is a better quality of generated FCA results, showing rules and behaviors which may not be as clear in an attempt to correlate by the conventional method.

Keywords: Many-Core Processor Architectures, Networks-on-Chip, Parallel Applications, Formal Concept Analysis, Full-System Simulation

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AFC – *Análise Formal de Conceitos*

FAST – *Fast Fourier Transform*

FN – *Friendly Numbers*

GF – *Gaussian Filter*

IS – *Integer Sort*

KM – *k-means*

LU – *Lower-Upper Gauss-Seidel*

NoCs – Redes-em-Chip, do inglês *Networks-on-Chip*

NPB – NAS *Parallel Benchmark*

PARSEC – *Princeton Application Repository for Shared-Memory Computers*

TSP – *Traveller-Salesman Problem*

SUMÁRIO

1 INTRODUÇÃO

Sistemas computacionais *many-core* são arquiteturas de computadores que possuem dezenas, centenas ou milhares de núcleos de processamento. A principal vantagem é o ganho de desempenho considerável devido ao alto poder de processamento paralelo e heterogeneidade. Bioinformática e Inteligencia Artificial [Daya et al., 2017], [Xue et al., 2014] são exemplos de algumas áreas que se beneficiam de sistemas *many-core*.

Esses sistemas possuem algumas características que os diferem dos sistemas convencionais* [Singh et al., 2013]. Uma dessas características, como exemplo, é a grande quantidade de processadores que o sistema possui. Este fator faz com que o consumo energético que o sistema possui passe a ter uma importância ainda maior no momento em que é projetado. A evolução de um sistema *many-core*† intra-*chip* depende da influência dos núcleos (processadores em *chip*), memória, interconexão, entre outros elementos. São eles que serão modificados para extrair o máximo de desempenho de uma arquitetura. Diferentes configurações de um sistema com os mesmos recursos afetam o ganho de tempo, consumo de energia, comunicação entre os núcleos, etc [Singh et al., 2013]. Sendo assim, é preciso tomar uma decisão para que os elementos de um sistema *many-core* intra-*chip* tenham o melhor desempenho.

A aplicação é a chave para a escolha de como o sistema será projetado e avaliado. Normalmente, consumo de energia e tempo de execução são os quesitos para esta escolha. Cada aplicação tem suas peculiaridades que aumentam ou diminuem o desempenho de uma execução de acordo com a configuração do sistema.

Com a diversidade de aplicações existentes, é praticamente impossível criar um sistema que possua desempenho melhor do que todos. Sendo assim, cada sistema é projetado para a aplicação ou aplicações que ele irá executar. Até mesmo os sistemas de propósito geral são moldados em um nível aceitável de resultados de acordo com *benchmarks*. A evolução de cada sistema é voltado também para a aplicação para o qual foi projetado.

Para que estes sistemas possam evoluir, avaliações são necessárias [Jain, 1990]. Investigando quais elementos mais influenciam no sistema é possível realizar modificações visando aumentar o desempenho. Saber analisar os resultados apresentados por um sistema ou um conjunto de sistemas é determinante para escolha de qual arquitetura irá ser projetada e/ou utilizada para executar uma determinada aplicação ou um conjunto de

*Sistemas que possuem um (*single-core*) ou poucos núcleos

†O termo *many-core* será usado no restante da dissertação no contexto em *chip*

aplicações. E essa analise não fica restrita ao desempenho.

A Tabela 1 mostra um conjunto de dados que podem ser analisados em um sistema computacional, bem como suas respectivas unidades de medida e se é dado de entrada (configuração do sistema) ou de saída (resultado de comportamento da execução).

Tabela 1 – Exemplos de variáveis de um sistema

Característica	Unidade de Medida	Tipo
Tempo	Segundos	Saída
Consumo de Potência	*Watts*	Saída
Consumo de Energia	*Joules*	Saída
Acesso a memória	Nº de acessos	Saída
Operações Realizadas	Nº de operações	Saída
Frequência dos Processadores	*Hertz*	Entrada
Número de processadores	Quantidade	Entrada
Capacidade de Memória Principal	*Bytes*	Entrada
Capacidade de Memória *cache*	*Bytes*	Entrada
Aplicação	Aplicação	Entrada

Fonte: Elaborado pelo Autor

Com essas variáveis, é possível realizar um estudo sobre o comportamento do sistema sobre dados de entrada. Assim, teria-se de analisar qual o desempenho (tempo), gasto energético (consumo de potência e energia) e fluxo de comunicação/processos (acesso a memória e número de operações realizadas). O tempo para análise seria proporcional ao número de valores que cada variável pode assumir.

Sistemas *many-core* em *chip* possuem um nível de complexidade maior se tratando de comunicação, disponibilidade de recursos, dentre outros fatores, se comparados a sistemas com menos um ou único núcleo. Assim, quanto mais a complexidade de um sistema aumenta, maiores são as possibilidades de resultados sobre eles, ficando cada vez mais difícil de serem analisados.

Devido a essa característica, a análise destes sistemas torna-se mais complexa [Diaz et al., 2012]. A quantidade de dados a serem analisados aumenta em relação a quantidade de núcleos do sistema. Não se pode analisar o sistema como um todo se o objetivo for uma análise mais completa.

Se a fase de análise for demorada, o seu processo de evolução pode ser mais demorado ainda, uma vez que depende dessa parte para ocorrer. Muitos métodos de análise são feitos apenas com base na experiência e base científica do analisador, seguindo os métodos baseados em experimentos, simulação ou modelagem analítica. A análise formal de conceitos auxiliaria na aceleração de todo esse processo.

Análise Formal de Conceitos (AFC, ou FCA: Formal Concept Analysis) é uma

técnica utilizada em estatística para geração de conceitos e regras que relacionam as características da análise. Assim, é possível gerar em pouco tempo, conhecimento a cerca das execuções realizadas.

A AFC analisa a recorrência de uma ou várias características em relação a todas as outras, ou seja, qual a porcentagem que atributos, sendo atributos, neste caso, os sistema em questão, aparecem em conjunto com variáveis, sendo variáveis as características, como bom desempenho, alto consumo de memória, entre outros. A recorrência que possui 100% de dado(s) de acontecimento em relação ao(s) dado(s) de objetivo são chamados de conceitos enquanto as que possuem menos que 100% são chamadas de regras.

Os conceitos mostram o que sempre ocorrerá no sistema se aquele conjunto de dados de acontecimento estiver presente na simulação. Já as regras mostram quais são as chances de determinado conjunto de dados de objetivo ocorrer caso um conjunto de dados de acontecimento estiver presentes.

Toda e qualquer característica do sistema pode ser dado de acontecimento ou de objetivo em conceitos ou regras diferentes. Assim, um determinado dado de acontecimento e um determinado dado de objetivo pode gerar um conceito, ou seja, 100% de recorrência, mas o contrário não necessariamente tem a mesma porcentagem.

Assim, uma análise que possui uma quantidade grande de características terá poucas chances de possuir muitos conceitos ou regras com uma alta porcentagem de recorrência. Porém, ainda assim essas regras podem mostrar quais são as características que mais influenciam não só no desempenho do sistema, mas também em outros dados relevantes.

A AFC acelera o processo de evolução do sistema, uma vez que pode apontar os indícios de onde desenvolver melhor as aplicações de forma a se extrair ao máximo as vantagens de um sistema. Assim, versões futuras das aplicações e até mesmo do próprio sistema podem surgir mais eficientes em menor tempo se este método for utilizado. Além disso, a AFC também aponta quais são as melhores configurações dos sistemas para cada aplicação.

Muitas vezes, o processo de evolução de um sistema, seja ele *many-core* ou não, fica atrelado ao conhecimento do responsável pela parte de análise dos dados coletados. O tempo de análise e/ou algumas falhas humanas, como falta de atenção ou experiência, podem prejudicar esse processo.

O intuito de se aplicar a AFC em análises de sistemas *many-core* é diminuir o tempo que se leva para analisar um sistema bem como reduzir a falha humana ao longo do processo. Com as regras e conceitos, é possível identificar quais características do sistema são mais relevantes e quais não são, agilizando o processo de identificação destas características e entender melhor os sistemas analisados.

1.1 Problema

Com a grande quantidade de variáveis que compõem um sistema *many-core*, avaliá-lo se torna um desafio. O processo de análise do sistema apenas pela experiência de um cientista ou qualquer outro avaliador pode ser demorado e haver descuidos em sua avaliação. Apenas o conhecimento do avaliador pode não ser suficiente e nesse sentido, abre-se uma lacuna em que torna-se necessário o uso de uma técnica para auxiliar na avaliação de sistemas *many-core*.

1.2 Objetivos

O objetivo dessa dissertação é apresentar o AFC como técnica de análise de execuções em sistemas *many-core* em *chip*, apresentando as regras e os conceitos gerados. Com ela, apontar quais características podem ser alteradas para aumentar o desempenho do sistema e quais são os melhores sistemas de cada aplicação. Assim, é possível dividir nos seguintes objetivos:

- Simular arquiteturas de sistemas *many-core* com diferentes topologias implementadas no simulador GEM5 [Binkert et al., 2011].
- Executar aplicações paralelas do *benchmark* CAP Bench [Souza et al., 2017] nas arquiteturas simuladas.
- Utilizar a AFC como técnica de avaliação nos resultados obtidos das execuções das aplicações.
- Comparar a avaliação feita pela AFC com a avaliação convencional.

1.3 Contribuição

A contribuição dessa dissertação está em apresentar a técnica AFC como uma alternativa viável para avaliação de desempenho. De acordo com a revisão do estado da arte, não foram encontrados trabalhos de pesquisa nessa direção.

1.4 Organização da Dissertação

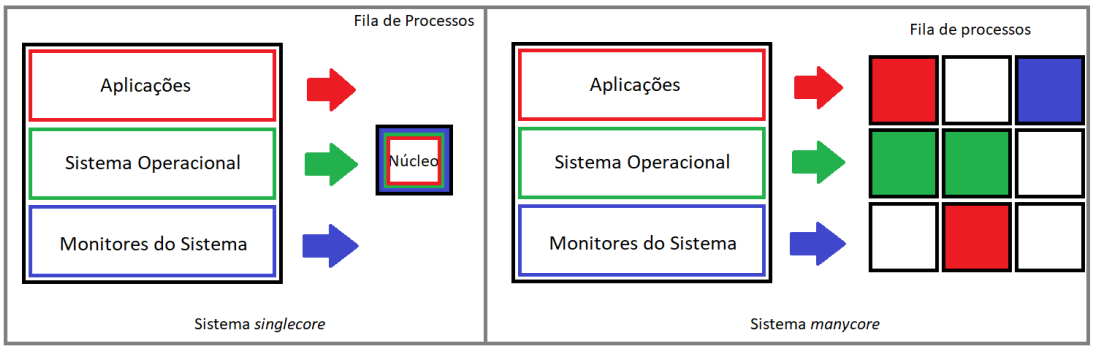
Este trabalho está divido em cinco capítulos. O segundo capítulo aborda o referencial teórico. O terceiro capítulo destina-se à metodologia utilizada. O quarto capítulo disserta sobre resultados obtidos. O quinto capítulo apresenta as conclusões.

2 REFERENCIAL TEÓRICO

Sistemas *many-core* em *chip* são arquiteturas de processadores que possuem dezenas, centenas ou milhares de núcleos. Apenas esse aspecto já faz com que estes sistemas sejam planejados de forma completamente diferente do que os sistemas computacionais convencionais.

São várias características diferentes entre o convencional e o *many-core*, entre elas poder individual de processamento dos núcleos, estrutura de programação, consumo de energia, disponibilidade de recursos e comunicação. A Figura 1 mostra um sistema convencional e um sistema *many-core*.

Figura 1 – Arquitetura Convencional *versus* Arquitetura *many-core*



Fonte: Elaborado pelo Autor

Os blocos à esquerda da imagem representam a fila de processos enquanto os blocos à direita representam os núcleos. Enquanto há apenas um núcleo processando todos os processos da arquitetura da esquerda há vários núcleos trabalhando em paralelo na arquitetura da direita.

Uma arquitetura *many-core* pode possuir os mais potentes processadores da atualidade. Porém, dado os custos, consumo de energia e dificuldade de integração entre eles, entre outros fatores, uma alternativa é, ao invés de se colocar um processador com alto desempenho, pode-se utilizar processadores com desempenhos razoáveis em seu lugar.

Ao se falar de algoritmos e aplicações que o sistema terá de executar, é válido ressaltar que nem sempre o processamento paralelo é a melhor solução. Cada aplicação tem sua própria natureza e é ela quem define qual caminho seguir a fim de se otimizar desempenho. Por exemplo, pode-se ter duas formas de ser ordenar números naturais: os algoritmos BubbleSort e BucketSort.

O primeiro algoritmo tem a premissa básica de sempre pegar um número e colocar ele na ordem correta dentro os números que já estão ordenados até não haver mais números. Isso faz com que o passo anterior precisa ser definido para que se execute o próximo. Há abordagens paralelas para esse algoritmo, como dividir os números em pequenos conjuntos, porém, ao final, sempre precisará juntar todos os números para as comparações serem feitas.

O segundo algoritmo cria uma lista de valores cuja posição do número sempre será igual ao seu valor. Sendo assim, o número 0 seria o primeiro da lista, o número 1 seria o segundo e assim sucessivamente. Pode-se perceber que não há dependência entre os passos desse algoritmo, podendo ser executados em paralelo.

Sendo assim, é preciso que os próprios algoritmos sejam elaborados para atingir um alto desempenho em sistemas *many-core*. Algoritmos com muitas iterações com baixas dependências entre si possuem um bom potencial de desempenho nesses sistemas, enquanto algoritmos com poucas iterações, recursividade e/ou alta dependência entre seus passos apresentam desempenho próximo ou pior do que suas respectivas versões paralelas.

A viabilidade do sistema também leva em consideração o seu consumo de energia. Com o crescimento do número de núcleos de sistema, cresce também o número de elementos necessários para seu funcionamento. E o crescimento do sistema é diretamente proporcional ao crescimento de seu consumo se utilizados os mesmos núcleos de processamento. Porém, o consumo de energia é menor se usado vários processadores de menor poder de processamento do que um processador mais potente, dependendo dos processadores utilizados.

À medida que o sistema aumenta, é preciso pensar na disponibilidade de seus recursos. Memória (principal e secundária) e outros elementos (como GPU's e aceleradores) têm de estar acessíveis a todos os processadores do sistema.

Em um exemplo prático, se um sistema possui uma centena de processadores buscando informações que não estão presentes em suas devidas memórias *cache*, todos terão de acessar a memória principal para obter estes dados. Então, haver um sistema com memórias distribuídas nesse caso pode acabar aliviando o fluxo de informações, diminuindo assim seu gargalo.

A comunicação para esses sistemas também precisa passar pela análise. Uma vez que não só processadores precisam comunicar entre si, bem como outros elementos, é preciso que o sistema seja adaptado para não haver degradação de desempenho no seu fluxo de dados. Sendo assim, a comunicação em sistemas *many-core* precisa ser pensada de forma a não degradar o desempenho do sistema enquanto os algoritmos são executados.

Arquiteturas com poucos núcleos adotam, geralmente, o barramento como prin-

cipal forma de comunicação entre seus núcleos. É uma solução eficaz e não apresenta consideráveis perdas de desempenho até que se atinja um certo número de núcleos. A comunicação aqui é bem simples. Os núcleos estabelecem comunicação quando o barramento está livre. Porém, quando a quantidade de processadores aumenta, é preciso trocar a abordagem para a comunicação não virar um gargalo.

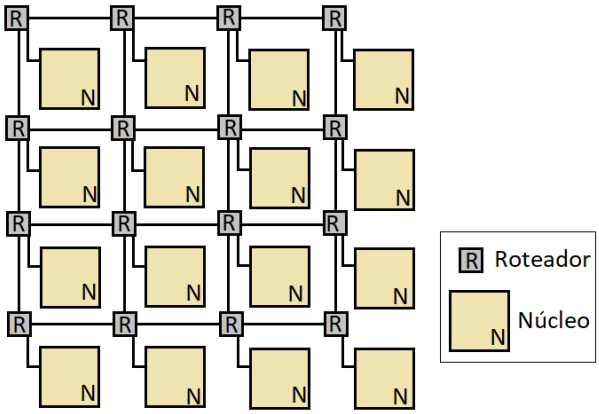
Dessa forma, inúmeras pesquisas [Wentzlaff and Agarwal, 2009] surgiram para que se desenvolvessem novas técnicas de comunicação entre os núcleos dentro do sistema. Dentre elas, existe a comunicação através de redes-em-*chip*, cujo foi utilizada nesse trabalho.

2.1 Redes-em-Chip

Formalmente, redes-em-chip (do inglês *Network-on-Chip* (Redes-em-Chip, do inglês *Networks-on-Chip* (NoCs))), é uma arquitetura em malha de roteadores onde recursos são colocados em interfaces ligadas a roteadores [Kumar et al., 2002].

Um exemplo foi desenhado como uma malha 2-D de roteadores e recursos provendo integração física em nível arquitetural. Cada roteador é conectado a um recurso e roteadores vizinhos e cada recurso é conectado a um roteador. Um recurso pode ser um processador, memória, FPGA, um hardware modificado ou qualquer outro elemento que cabe a um encaixe de roteador e cumpre funções dentro da interface da rede-em-chip. A Figura 2 mostra uma arquitetura de rede-em-chip.

Figura 2 – Rede em Chip



Fonte: Elaborado pelo Autor

Uma característica que deve ser analisada em redes-em-chip são os algoritmos de roteamento. É dever de cada roteador encaminhar os dados de um lugar para outro, seja processador ou memória. Se por um lado a comunicação entre os elementos ficam simplificadas, já que eles têm de saber apenas com qual elemento iram se comunicar, cabe aos roteadores dizer como.

Para tanto, os roteadores necessitam do algoritmo de roteamento para calcular qual a melhor rota um determinado fluxo de dados deve seguir. Muitas vezes um roteador receberá e encaminhará um pacote de dados sem que faça nenhum processamento dos dados em si. Isso ocorre, pois, seu pacote de dados não chegou ao destino final. Ao chegar ao destino final, será encaminhado para o processador ou memória afixado àquele roteador.

A principal vantagem das redes-em-chip está na forma simples em que os roteadores se conectam. O grande desafio dos sistemas *many-core* é apresentar uma comunicação eficiente entre seus núcleos. Com essas redes, pode-se criar inúmeras formas de comunicação entre os processadores focando apenas nos algoritmos de roteamento para ditar o comportamento do fluxo de dados dentro da rede. Acertar um bom algoritmo de roteamento com uma boa organização dos elementos, o desempenho de um sistema pode ser consideravelmente aumentado.

Com o passar dos anos, essas redes evoluíram e estão aparecendo cada vez mais em estudos dos mais variados. Como exemplos, essa abordagem já foi aplicada em redes neurais artificiais [Theocharides et al., 2004], aprendizado de máquina [Qian et al., 2016] e novas arquiteturas de processadores [Daya et al., 2017]. Para este trabalho, as redes-em-chip foram utilizadas para realizarem a comunicação entre os processadores de cada arquitetura.

2.1.1 Topologias de Redes-em-Chip

Topologias, no contexto de redes-em-chip, dizem respeito à forma como os recursos e elementos estão conectados [Kumar et al., 2002]. Conforme descrito, redes-em-chip foram desenvolvidas em diversas topologias existentes que podem auxiliar na comunicação entre os núcleos de um sistema *many-core*, como por exemplo as topologias *torus* e *cluster*.

A topologia em barramento consiste em vários elementos interligados por um único canal de comunicação. Sendo assim, dois elementos podem se comunicarem por vez.

A topologia em malha é basicamente uma rede 2-D cujo roteadores estão organizados em n colunas e m linhas e cada roteador se comunica diretamente com seus vizinhos das linhas ou colunas subsequentes.

A topologia *torus* se assemelha a topologia em malha, com a diferença que em seus extremos, ou seja, as primeiras e últimas linhas e colunas são interligadas entre si. Sendo assim, em uma malha com n colunas e m linhas, os roteadores da coluna 1 se comunicam diretamente com a coluna N e os roteadores da linha 1 se comunicam diretamente com a linha M.

A topologia estrela apresenta um roteador central e todos os outros roteadores

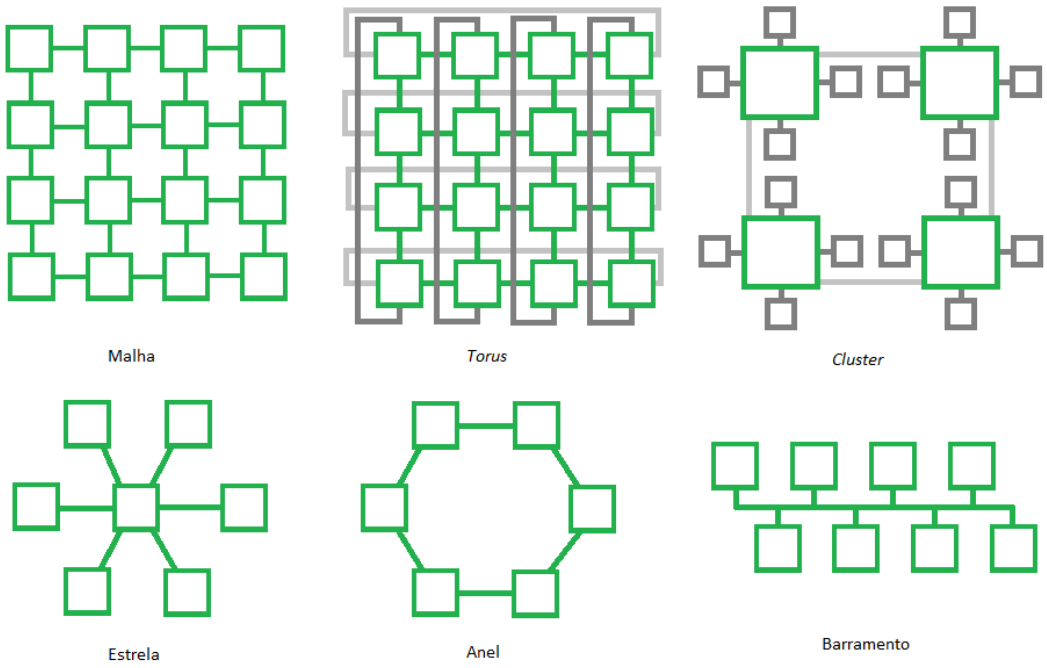
da rede se conectam a este roteador. Assim, a troca de informações se intensifica neste roteador mas a comunicação fica simplificada, pois cada núcleo fica a, no máximo, três roteadores de distância (o próprio, o central e o roteador de destino).

A topologia em anel são roteadores que são ligados à e somente a outros dois roteadores e outros elementos da rede. Este nome se da pelo fato de a topologia parecer ser um círculo de roteadores.

A topologia *cluster* se pode dizer que seja uma topologia híbrida entre a topologia em malha e a topologia em estrela. Uma malha de roteadores é feita enquanto cada roteador dessa malha possui outros roteadores ligados apenas a ele.

A malha original foi utilizada, mas também foram desenvolvidas as topologias *torus* e *cluster*. A Figura 3 mostra os seis tipos de topologias de redes-em-chip utilizadas nesse trabalho.

Figura 3 – Exemplos de Topologias



Fonte: Elaborada pelo Autor

Nota-se que na topologia *cluster*, há roteadores que possuem apenas duas conexões, sendo utilizados para interligar processadores à rede enquanto os roteadores que possuem conexão com outros quatro roteadores sendo utilizados para interligar memórias a rede. Nas outras topologias, todos os roteadores seguem o mesmo padrão, conectando-se a outros quatro roteadores, podendo haver uma memória ou um processador, com exceção feita as topologias em malha cujo roteadores extremos, ou seja, aqueles que estão nas extremidades da topologia, possuem conexão com outros dois ou três roteadores.

Com a rede-em-chip, fica a cargo dos roteadores fazer o envio e recebendo de dados dentro da rede, através de algoritmos de roteamento, para que haja comunicação entre os processadores e outros componentes dentro do sistema.

Havendo tantas topologias diferentes, com consumos de energia e quantidade de elementos diferentes, entre outras características, é preciso decidir qual delas se utilizar em um sistema *many-core*. A utilização de *benchmarks* pode auxiliar na escolha da topologia.

2.2 *Benchmarks* Paralelos

Benchmark é um conjunto de aplicações executadas em um determinado sistema com o intuito de se avaliar o seu desempenho [Hockney et al., 1996]. Com eles, é possível comparar uma arquitetura a outra em vários aspectos, como consumo de energia, tempo de execução, utilização de recursos, dentre outras coisas. Após a execução, são gerados traços de execução (*trace logs*) que possuem os dados da execução das aplicações.

Os *benchrmarks* são utilizados para se ter uma forma de avaliação do sistema como um todo, podendo ser executados tanto em sistemas reais como em sistemas simulados. Com a evolução dos computadores, ter uma forma de se avaliar o sistema de maneira a poder compará-lo com outros é importante.

Para a avaliação de sistemas *many-core*, utiliza-se *benchmarks* paralelos. Isso implica em aplicações criadas para serem executadas em paralelo, e não na sua forma serial.

O *Nas Parallel Benchmark*, ou NAS *Parallel Benchmark* (NPB), é um *benchmark* paralelo desenvolvido pela NASA com o intuito de se avaliar o desempenho de supercomputadores paralelos [Bailey et al., 1991]. O NPB conta originalmente com oito aplicações.

O *Princeton Application Repository for Shared-Memory Computers* (PARSEC), é um *benchmark* composto por programas *multithread*. O conjunto de aplicações foca em cargas de trabalho muito utilizadas e foi projetado como uma representação da próxima geração de programas de memória compartilhada para *chips* e sistemas *many-core* [Bienia et al., 2008].

O CAPBench é um *benchmark* para avaliação de desempenho e consumo energético de processadores *many-core* de baixo consumo. Este *benchmark* oferece um diversificado conjunto de aplicações em relação a padrões paralelos, tipos de cargas de trabalho, comunicação, estratégias de carga de intensidade, adequadas para uma ampla compreensão do desempenho e consumo de energia do sistema [Souza et al., 2017].

Cada *benchmark* tem o intuito de avaliar um ou um conjunto de medidas. Suas versões paralelas têm em comum o objetivo de avaliar o desempenho exclusivamente para arquiteturas paralelas ou *multithread*. Com os arquivos de traços, pode-se avaliar as

arquiteturas usando alguma técnica de avaliação.

2.3 Métodos de Avaliação para Arquiteturas *many-core*

O que define um método de avaliação é a junção da técnica utilizada em conjunto com suas métricas. Para se definir as métricas, primeiro é preciso definir a técnica, uma vez que algumas métricas podem ser difíceis de se conseguir com algumas técnicas.

Existem três tipos de técnicas de avaliação. São elas: modelagem analítica, simulações e medição [Jain, 1990], como estágio de vida, ferramentas, custos, tempo necessário ou acurácia. Na Tabela 2, pode-se ver alguns desses fatores com algumas considerações, em ordem de importância, do primeiro a último, na hora de se escolher uma técnica.

Tabela 2 – Técnicas de Modelagem

Critério	Modelagem Analítica	Simulação	Medição
Estágio	Qualquer um	Qualquer um	Protótipo
Tempo Necessário	Pequeno	Médio	Variável
Ferramentas	Analistas	Linguagens de Programação	Instrumentação
Acurácia	Baixa	Média	Variável
Custo	Baixo	Médio	Alto

Fonte: Elaborado pelo Autor

A chave para escolha de uma técnica de avaliação é o estágio do ciclo de vida que o sistema se encontra. Para medições, é preciso que algo similar ao sistema proposto já exista, para desenvolver uma versão melhorada do produto. Se for um novo conceito, modelos analíticos e simulações são as únicas técnicas que podem ser escolhidas. Estes últimas duas técnicas podem ser usadas onde a medição não é possível, porém serão mais convincentes se forem feitas com bases em medições prévias.

Seguindo as técnicas, temos o tempo necessário para avaliação. Se o tempo for curto, modelagem analítica pode ser a única saída. Caso contrário, utilizar alguma outra técnica pode ser mais vantajoso.

As ferramentas necessárias são a próxima consideração. Basicamente, o que a modelagem analítica precisa é de um ou vários analístas. Já as simulações necessitam de conhecimentos em linguagens de programação, que não é tão simples. Para medição, instrumentos de medição são necessários.

A acurácia pode ser um problema caso seja escolhida a modelagem analítica, visto que é, das três técnicas, a mais distante do sistema real. Simulações tendem a ser melhores uma vez que incorporam maiores detalhes do sistema. A medição, mesmo parecendo como certeza de uma acurácia consideravelmente alta, leva em consideração as variáveis

de ambiente do sistema, como configuração do sistema, tipo de carga de trabalho e tempo de medida, podendo todos esses dados serem únicos para o experimento.

O custo pode ser determinante na escolha se as opções de investimento não forem "ambiciosas". Com baixo orçamento, modelagem soa como a melhor opção. Porém, com um bom aporte, qualquer uma das três técnicas se tornam escolhas viáveis.

Definido a técnica, agora é preciso definir quais métricas serão utilizadas no avaliação de desempenho. Uma forma de se definir este conjunto de métricas é listar quais são os tipos de serviços oferecidos pelo sistema. Em sistemas *many-core*, por exemplo, o que se busca é o ganho de desempenho em relação ao tempo de execução. Métricas que se relacionam com este parâmetro são uma boa escolha.

Os possíveis valores de métricas se encaixam em três categorias, normalmente. O sistema pode gerar as possíveis saídas após executar o serviço corretamente, incorretamente ou se recusar a efetuar o serviço.

Se o sistema executou o serviço corretamente, o desempenho é medido através do tempo em que o sistema utilizou, juntamente com os recursos consumidos para ser executado. Se o sistema executou o serviço incorretamente, classificar os erros e determinar em quais circunstâncias ele pode ocorrer é importante. Se o sistema recusou-se a executar o serviço, é preciso descobrir quais são as limitações que o sistema impõe para execução.

Feitas as escolhas, é possível prosseguir com as avaliações. Os métodos propostos são relativamente simples e constituem na escolha da técnica, da métrica e assim sua análise. Muitas das avaliações feitas tendem a ser a comparação de resultados colhidos e variados estudos afim de se montar um quadro de resultados com base nessas comparações.

Este trabalho se propõe a apresentar um método de análise simples, eficaz e que elimina a necessidade de um bom conhecimento prévio sobre o contexto de sistemas *many-core*, através da utilização de Análise Formal de Conceitos (*Análise Formal de Conceitos* (AFC)).

2.4 Análise Formal de Conceitos

Análise formal de Conceitos (ou *Formal Concept Analisys*) é uma técnica utilizada em Ciência da Computação para formação de conjuntos de objetos com características em comum que levam a propriedades em comum [Dias and Vieria, 2011]. Este método é utilizado para a formação de conceitos a cerca de uma base de objetos.

A motivação original da análise de conceitos formais seria a descoberta de conhecimentos sobre bases de dados complexa o suficiente para não ser perceptível, uma vez que os conhecimentos óbvios por si só são relativamente fáceis de serem descobertos.

Sendo assim, o intuito é gerar uma representação das propriedades dessas bases através de conceitos formais.

Um conceito formal é uma tripla (G, M, I), em que G é um conjunto cujos elementos são denominados objetos, M é um conjunto cujos membros são chamados atributos e $I \supset GxM$ é uma relação denominada relação de incidência [Dias and Vieria, 2011]. Atributos são os dados enquanto objetivos, ou variáveis, são as métricas. Como exemplo, podemos ter um conjunto de polígonos onde se teria vários tipos de quadrados, triângulos e círculos como atributos e número de lados, área e comprimento como variáveis.

Se $(g, m) \in I$, diz-se que "o objeto g tem o atributo m". Um contexto formal é representado por uma tabela em que os objetos aparecem nos cabeçalhos das linhas enquanto os atributos aparecem nos cabeçalhos das colunas. A interseção das linhas com as colunas dizem se um determinado objeto possui determinado atributo. A Tabela 3 mostra uma tabela de um contexto.

Tabela 3 – Tabela de Contexto Formal

	Objeto 1	Objeto 2	Objeto 3	Objeto 4	Objeto 5
Atributo A	X				X
Atributo B		X	X		
Atributo C		X	X		X
Atributo D	X			X	
Atributo E		X			X

Fonte: Elaborado pelo Autor

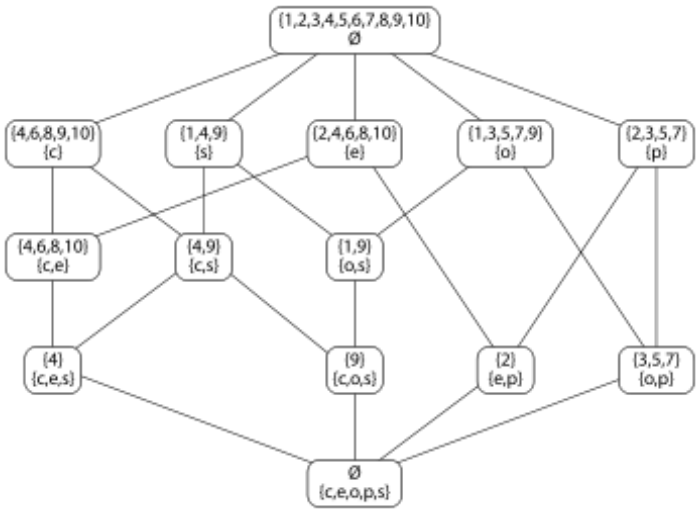
Dado um conjunto de objetos $A \supset G$ e um contexto formal (G,M,I), pode-se perguntar que atributos em M são comuns a todos os objetos de A. Similarmente, pode-se perguntar, para um conjunto $B \supset M$, que têm todos os atributos de B. Tais argumentos são respondidos pelos operadores de derivação, assim definidos: $A' = m \in M | \forall g \in A(g, m) \in I$ e $B' = g \in G | \forall m \in B(g, m) \in I$.

Um conceito formal é um par $(A, B) \in GxM$ tal que $A' = B$ e $B' = A$, onde A é denominado extensão e B a intenção do conceito. O conjunto dos conceitos formais é ordenado pela ordem parcial $\leq$ tal qual para quaisquer dois conceitos formais (A1, B1) e (A2, B2), $(A1, B1) \leq (A2, B2)$ se e somente se $A1 \supset A2$ (e, neste caso $B2 \supset B1$). O conjunto de conceitos ordenados por $\leq$ constitui em um reticulado completo chamado reticulado conceitual. O reticulado conceitual obtido a partir e um contexo formal (G,M,I) é denotado por $\beta(G, M, I)$.

O teorema básico sobre reticulados conceituais diz que um reticulado conceitual (beta)(G, M, I) é um reticulado completo no qual para qualquer conjunto $C \supset \beta(G, M, I)$ o *supremum* e o *infinitum* são dados por $\delta C = (TX, (SY)00)$ e W C = ((S X) 00 , T Y), sendo $X = A|(A, B) \in CeY = B|(A, B) \in C$ [Dias and Vieria, 2011]. A baixo, a Figura

4 mostra um reticulado completo.

Figura 4 – exemplo de reticulado completo



Fonte: Elaborado pelo Autor

Na Figura 4, seja o reticulado (beta)(G,M,I), onde:

- G = 1,2,3,4,5,6,7,8,9,10
- M = c,e,o,p,s
- I = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10,), (4, 6, 8, 9, 10, c), (1, 4, 9, s), (2, 4, 6, 8, 10, e), (3, 5, 7, 9, o), (2, 3, 5, 7, p), (4, 6, 8, 10, c, e), (4, 9, c, s), (1, 9, o, s), (4, c, e, s), (9, c, o, s), (2, e, p), (3, 5, 7, o, p), (, c, e, o, p, s)

Os números representam os objetos enquanto as letras representam os atributos. Os elementos se ligam de acordo com as suas interseções. Cada nó representa um conceito gerado. No extremo superior, tende-se a ter conceitos com uma quantidade de objetos maiores enquanto no extremo inferior tende-se a ter conceitos com uma quantidade de atributos maiores. Ao final, temos o conjunto I que representa todos os conceitos gerados.

Algo que a AFC também gera são as regras de associação. Pode-se dizer que estas regras são conceitos que não necessariamente 100% de seus objetos possuem 100% de seus atributos. Essa porcentagem é chamada de confiança. Regras de associação são descobertas onde se e somente se 100% de seus objetos possuírem cada um 100% de seus atributos.

Por exemplo, para um dos elementos do conjunto, a tupla (4, 9, c, s), se for acrescentar o objeto 3 ao conjunto de objetos, o conceito passa a ser somente uma regra

de associação, representada pela tupla (3, 4, 9, c, s). Essa regra possui uma confiança de 66,7%, aproximadamente, já que apenas 66,7% de seus elementos (4 e 9) possuem os todos os atributos (c e s).

Outro fator também analisado pela AFC é o suporte de cada regra de associação. Suporte é a razão entre a quantidade de elementos presentes dentro de um conjunto objetos de uma determinada regra e a quantidade de objetos do conjunto global. Utilizando novamente a regra do exemplo anterior, seja a tupla (3, 4, 9, c, s). Esta regra possui 30% de suporte, uma vez que possui três objetos, enquanto o conjunto universo possui dez.

Concluindo, a AFC é utilizado para gerar os conceitos e as regras de um determinado conjunto de objetos e atributos. Os conceitos, que possuem 100% de confiança, indicam uma forte tendência de comportamento sobre o que está sendo analisado. Regras com confiança alta também podem indicar comportamentos, porém com menos expressividade que os conceitos, uma vez que apresentam objetos que são exceções a regra.

Aplicado para a avaliação de sistemas *many-core*, podemos colocar inúmeros atributos a cerca das arquiteturas e/ou aplicações que se utiliza como base, uma vez que os arquivos de traço estejam prontos. Porém, a única restrição imposta pela AFC é que seus dados sejam binários, ou seja, os atributos não podem ser numéricos ou categóricos. Seja esse o caso, é preciso que uma binarização dos dados seja feita.

Por outro lado, a utilização da AFC mostra que não é preciso um conhecimento prévio sobre a área para a análise ser feita. Uma vez gerada as regras e os conceitos, é possível extrair o conhecimento a cerca dos objetos e atributos.

Sendo assim, a utilização da AFC para avaliação de arquiteturas *many-core* pode mostrar onde o sistema deve ser modificado com o objetivo de aumentar seu desempenho, bem como elementos que pouco/não influencia o sistema.

2.5 Trabalhos Correlatos

Não foi encontrado nenhum trabalho cuja AFC foi utilizada para se avaliar sistemas *many-core*. Há alguns trabalhos que utilizam o GEM5, tal como trabalho prévio dessa dissertação [Carmo et al., 2016], para avaliar estes sistemas e outros métodos de avaliação para sistemas *many-core*.

Em [Sun et al., 2002], arquiteturas organizadas por redes-em-chip foram prototipadas utilizando um simulador (ns-2) e então foram analisadas. Após as análises feitas, foi apresentado uma proposta de arquitetura com base nos resultados obtidos.

Em [Gratz et al., 2006], arquiteturas organizadas em redes-em-chip são analisadas em cargas realistas. Neste trabalho, os autores afirmam que é preciso mais do que *bench-*

marks sintéticos para a avaliação poder ser feita, juntamente com um protótipo intitulado TRIPS. Novamente, a avaliação convencional é utilizada para as avaliações, havendo descoberto que aumentar a largura de banda não afeta de forma significativa o desempenho do sistema.

Em [Lee et al., 2007], é feita uma análise de algumas topologias tradicionais (barramento, *peer-to-peer*) com redes-em-chip. A avaliação foi feita em uma aplicação de multimidia (*MPEG-2 encoder*) em um protótipo utilizando FPGA. A avaliação foi feita de forma convencional e mostrou que as arquiteturas organizadas por redes-em-chip escalaram bem se tratando de desempenho e consumo de energia.

Em [Butko et al., 2012], é feito um estudo para se avaliar a precisão do simulador GEM5. Os resultados mostraram que a acurácia varia de 1,39% à 17,94% dependendo do tráfego de memória do sistema simulado.

Em [Van Laer et al., 2013], utilizou-se GEM5 para simular um sistema computacional completo. Assim, através de uma rede de interconexão utilizando *crossbar*, foram mostradas melhorias que podem se fazer no processador simulado.

Em [Madalozzo et al., 2016], o foco é a na análise de impacto que a memória possui em arquiteturas *many-core*. É feita uma comparação entre sistemas com memória compartilhada e memórias distribuídas e a conclusão é que sistemas *many-core* com memória compartilhada possuem um gargalo de comunicação em torno da memória.

Em [Solanki et al., 2016], um estudo sobre como algoritmos de roteamento utilizados por redes-em-chip podem impactar no desempenho e no consumo de energia do sistema, levando em consideração o aumento do número de processadores nessas redes ao longo dos anos.

Em [Souza et al., 2017], arquiteturas são simuladas utilizando o GEM5 a fim de se comparar o consumo de energia entre sistemas *multi-core* que utilizam *clusters* (processamento paralelo) e sistemas *multi-core* com execuções individuais. Os resultados mostraram que aplicações com cargas regulares apresentam um consumo de energia menor em sistemas de *clusters multi-core*, enquanto sistemas com processamento individual se saem melhores em aplicações de cargas irregulares.

Em [Amor et al., 2017], é proposta uma nova organização de hierarquia de memória com o intuito de maximizar a utilização da memória disponível em cada nível de *cache*, assim como reduzir o tempo de acesso a memória evitando a migração dos dados. Utilizando o simulador GEM5 e o *benchmark* Splash2, os resultados mostraram uma redução de 24% na taxa de *cache miss* e uma melhora de 25% na latência da rede.

Em [Bhattacharya et al., 2017], uma metodologia de análise de desempenho para modelagem de rede-em-chip *single-cycle multi-hop asynchronous repeated traversal* (SMART)

é proposta que habilita pacotes que ignoram, parcialmente ou completamente, o caminho entre os roteadores. Este método foi analisado em dois simuladores, o GEM5 e o simulador de rede GARNET.

Todos os trabalhos apresentados nesta seção apresentam arquiteturas organizadas redes-em-chip e análises feitas para melhorar o desempenho destes sistemas. Todos utilizaram-se da análise convencional para atingir este objetivo. Com a aplicação da AFC, a fase de análise nestes trabalhos teria sido encurtada, uma vez que os indícios encontrados por este método mostram o que influencia o desempenho do sistema.

3 METODOLOGIA

Para se poder avaliar a AFC como método de análise de resultados, foi preciso aplicá-la a dados coletados de execuções de aplicações em sistemas computacionais. Para tanto, várias arquiteturas foram elaboradas para a obtenção de uma base de dados aceitável para as análises. Com o simulador GEM5 e o *benchmark* CAP Bench, simulações foram feitas para a geração dos dados. Estes também foram avaliados sem o uso da AFC a título de comparação. Este capítulo detalha o procedimento feito.

A Metodologia está separada em quatro seções. A primeira seção aborda as arquiteturas e aplicações utilizadas como base deste trabalho. A segunda seção apresenta o simulador GEM 5. A terceira seção é sobre os formas de avaliação dos resultados separados por arquitetura e por aplicação. Na quarta seção tem-se a avaliação geral.

3.1 Arquiteturas e Aplicações Utilizadas

Esta seção é destinada à apresentação das arquiteturas e aplicações usadas. Ao todo, foram elaboradas 30 arquiteturas, variando em quantidade de núcleos, capacidade de memória *cache* e organização.

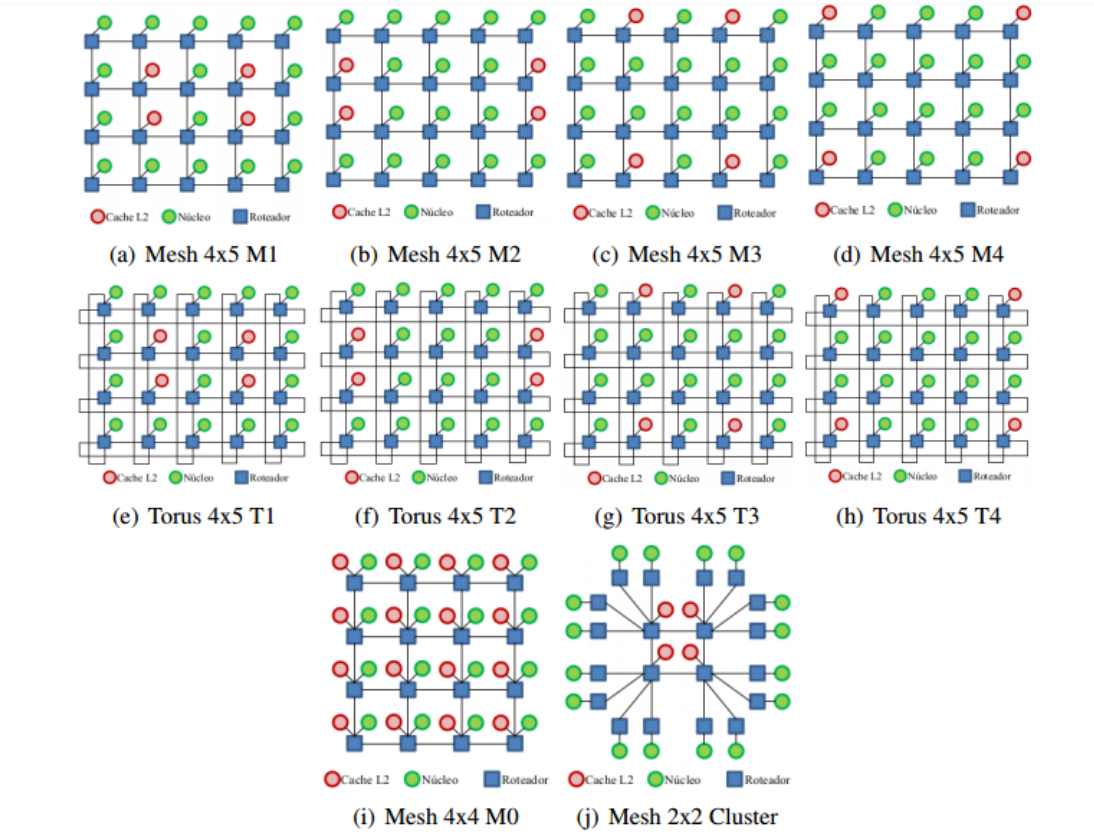
A quantidade de núcleos varia entre 16, 32 e 64 núcleos. O intuito deste parâmetro é descobrir qual o impacto que o aumento de núcleos tem nos resultados finais, sejam eles positivos, negativos ou neutros.

A capacidade de memória *cache* varia entre 16 kB e 64 kB de memória. Variar o tamanho da memória disponível para os núcleos implica em saber se o desempenho será alterado de acordo com esse parâmetro.

Para se poder avaliar o impacto de desempenho provocado pela forma cujos processadores estão dispostos, foram escolhidas três tipos de topologias. São elas: malha, *torus* e *cluster*. As Figuras 1(a), 1(e) e 1(j) mostram os tipos de organização, respectivamente.

A Figura 5 mostra exemplos de arquiteturas com 16 núcleos que foram utilizadas nestes trabalho. Todas as outras derivam destes exemplos, duplicando a estrutura (nos casos de 32 núcleos) ou quadruplicando (no caso de 64 núcleos).

Figura 5 – Exemplos das arquiteturas utilizadas com 16 núcleos



Fonte: [Souza, 2015]

As aplicações utilizadas neste trabalho estão presentes no CAPBench [Souza et al., 2017]. É um conjunto de *benchmarks* desenvolvidos originalmente para uma arquitetura específica de um sistema composto unicamente por processadores como nós de processamento (MPPA-256). Um diferencial desta ferramenta está no fato que, além de desempenho, ela também analisa consumo de energia da arquitetura.

Este *benchmark* possui as aplicações *Fast Fourier Transform* (FAST) (Features from Accelerated Segment Test), *Friendly Numbers* (FN) (Friendly Numbers), *Gaussian Filter* (GF) (Gaussian Filter), *Integer Sort* (IS) (Integer Sort), *k-means* (KM) (K-Means), *Lower-Upper Gauss-Seidel* (LU) (LU Factorization) e *Traveller-Salesman Problem* (TSP) (Traveling-Salesman Problem).

3.1.1 FAST

O FAST é um algoritmo proposto para identificar regiões de interesse em uma imagem [Goh et al., 2009]. Este algoritmo possui aplicações em aprendizado de máquina, reconhecimento de objetos e rastreamento [Bleser and Stricker, 2009].

O paralelismo presente neste algoritmo é bem claro, uma vez que pode-se analisar cada pixel independentemente dos outros.

3.1.2 FN

Na teoria dos números, números amigáveis (FN) são pares de números onde um número é igual a soma dos divisores do outro [Rolf, 1967].

O algoritmo de números amigáveis tem como objetivo encontrar um conjunto de pares de números naturais cujo um número qualquer do par seja a soma de todos os divisores do outro e vice versa. Assim como o FAST, o paralelismo aqui presente também é claro, uma vez que podemos analisar todas as possibilidades de pares em paralelo.

3.1.3 GF

Filtro Gaussiano e um filtro de suavização de imagens que desconsidera arestas (*pixels* vizinhos de valores bem distintos) em uma imagem. Seu comportamento é parecido aos filtros passa-baixa, onde o fator sigma influencia no quanto a imagem será suavizada. O filtro gaussiano possui diversas aplicações e implementações, como, por exemplo, redução de ruído [Deng and Cahill, 1993].

A ideia por trás de um filtro Gaussiano em uma distribuição 2-D é de uma função de expansão a partir de um ponto, implementado através de uma convolução. Uma convolução consiste em multiplicar par a par os elementos do filtro a cada pixel da imagem. A cada nova iteração o centro do filtro é deslocado em um pixel, geralmente da esquerda para a direita. Primeiro a convolução é aplicada ao eixo x e em seguida ao eixo y de cima para baixo.

No caso da implementação paralela, várias máscaras do filtro são lançadas ao mesmo tempo. Cada região é calculada separadamente e depois toda a imagem é unificada.

3.1.4 IS

A ordenação de inteiros não é um algoritmo em si, mas um problema genérico. Há diversas aplicações que implementam algum tipo de ordenação, como o *quicksort* ou *mergesort*. Neste caso, foi escolhido o *bucket sort*.

Este algoritmo possui o padrão de divisão e conquista. A implementação paralela divide o *bucket* em pequenos grupos de valores de forma que os valores são mapeados para os locais apropriados e então são colocados nos devidos grupos. Quando o grupo atinge um determinado número, é esvaziado e contabilizado.

3.1.5 KM

O *k-means* é uma estratégia de agrupamentos de itens através de suas similaridades. Elementos com características mais próximas tendem a ficarem em um mesmo grupo enquanto itens de características distintas tendem a ficar em grupos separados. Proposto em 1967, o algoritmo já possui diversas versões e várias aplicações, como na área de aprendizado em banco de dados [Wagstaff et al., 2001], analise de imagens [Ray and Turi, 1999] e *e-commerce* [Kim and Ahn, 2008].

Há várias implementações para o algoritmo. O escolhido para o *benchmark* foi o que utiliza centróides como representantes de cada grupo. No inicio da execução, são gerados k centróides aleatórios. É calculada a distância de cada item para cada centroide, o colocando no grupo do centróide mais próximo. Assim, são calculados os novos centróides baseados nas médias dos grupos. Se os centroides antigos forem diferentes dos novos, é feito todos os cálculos de distância novamente, realocando os itens em novos grupos. Se forem iguais, então o algoritmo se encerra.

A abordagem paralela é feita calculando as distâncias de cada centroide independente dos outros para cada item do conjunto.

3.1.6 LU

A fatoração LU é uma forma de fatoração de uma matriz não singular como produto de uma matriz inferior (Lower) e uma matriz superior (upper). Normalmente, sistemas computacionais resolvem sistemas de equações lineares e cálculo de matrizes inversas com esta técnica. Existem diversas abordagens para a sua implementação.

3.1.7 TSP

O problema do caixeiro viajante é conhecido tradicionalmente como a busca do menor caminho para se percorrer todas as cidades envolvidas no planejamento e retornar a origem. É um problema conhecido na teoria de grafos, onde cada cidade é um vértice e os caminhos representam as arestas.

Uma boa abordagem paralela a este algoritmo seria a implementação da força bruta. Cada núcleo pode calcular e armazenar qual o custo de cada caminho escolhido ao longo da execução. Ao final, os resultados seriam colocados juntos e o menor deles seria a resposta para o problema. Mas ainda assim demandaria muito tempo, dependendo da quantidade de núcleos. A implementação de heurísticas iriam minimizar consideravelmente o tempo de execução. No caso do CAP Bench, o menor caminho é atualizado sempre para toda a rede de núcleos de processamento, fazendo com que iterações exce-

dentes sejam interrompidas e se inicie uma nova.

O problema do caixeiro viajante é caracterizado por seu potencial paralelismo, uma vez que uma iteração é independente da outra e baixo acesso a memória.

3.2 Simulador GEM5

Como as arquiteturas utilizadas neste trabalho são baseadas na dissertação [Souza, 2015], o simulador escolhido foi o GEM5 para simular arquiteturas *many-core* afim de se avaliar o comportamento destes sistemas.

O simulador GEM5 é uma plataforma modular para pesquisas em arquiteturas de sistemas computacionais, abrangendo arquitetura em nível de sistema assim como a micro arquitetura do processador [Binkert et al., 2011]. Este simulador foi desenvolvido com o auxílio de várias instituições, incluindo a *National Science Foundation*, AMD, ARM, IBM, Intel, MIPS e SUN.

Entre as considerações feitas sobre este simulador e análises feitas por [Souza, 2015], destaca-se o fato de ser uma ferramenta de código aberto, possui uma boa documentação disponível para consulta e sua comunidade online é vasta e bastante ativa, tornando-se uma boa escolha de simulador.

Dentre as características da ferramenta, duas se mostram de grande importância para este trabalho: o fato de ser um simulador de sistema completo, ou seja, além da CPU, possui sistema detalhado de memória, baseado em eventos, *crossbars* e um modelo de controlador DRAM compatível com o modelo de memórias atuais; e um modelo de CPU baseado em rastreamento que captura os registros de tempo e consumo de potência.

Com esses dados, é possível não só avaliar o desempenho (o tempo) e gastos energéticos (o consumo de potência), mas também o comportamento da memória, como taxa de *cache miss*, número de acessos a memória e outros valores. E esses são, basicamente, os valores utilizados para a análise de resultados.

Para a simulação, é necessário o fornecimento de alguns dados para o simulador. São eles: número de núcleos a ser utilizados, capacidade da memória *cache* L2, tipo de arquitetura e número de *clusters*.

Para este trabalho, o número de núcleos varia entre 16, 32 ou 64. A capacidade de *cache* vária entre 16 e 64 kB de memória. O tipo de arquitetura pode ser malha, *torus* ou *cluster*. O número de *clusters* depende da quantidade de núcleos e qual a arquitetura escolhida.

3.2.1 Escalabilidade

Todas as simulações deste trabalho foram feitas através de *strongscale*. Isso quer dizer que o tamanho das cargas de trabalho para todas as aplicações foi o mesmo, variando número de processadores, capacidade de memória *cache* e outros parâmetros a fim de se medir o impacto no desempenho sem alterar o tamanho da carga.

O CAP Bench disponibiliza cinco tamanhos de cargas de trabalho por aplicação. São eles: *tiny*, *small*, *default*, *large* e *huge*.

- *tiny* - Cargas de trabalho muito pequenas, normalmente utilizadas para testar o comportamento das aplicações.
- *small* - Cargas de trabalho pequenas, para execuções rápidas das aplicações.
- *default* - Cargas de trabalho médias, com tamanhos típicos cujas aplicações costumam executar.
- *large* - Cargas de trabalho grande, que possuem um tamanho maior do que a média de cargas normais para essas aplicações.
- *huge* - Cargas de trabalho muito grande, utilizadas para uma análise mais profunda do processamento e desempenho da rede-em-chip.

O tamanho escolhido para testes foi o *default*. O foco deste trabalho é fazer análises das arquiteturas com aplicações reais. Assim, a utilização de cargas de trabalho mais parecidas com a realidade foi definitivo para esta escolha. Além deste fato, cargas muito grandes poderiam demandar um tempo grande de execução por parte dos simuladores. O que se quis provar foi a capacidade que a AFC tem de auxiliar na descoberta de conhecimento a cerca de avaliações de arquiteturas, sendo desnecessário cargas muito grandes. Todavia, o uso de cargas muito pequenas poderiam esconder características ou naturezas das aplicações escolhidas para simulação, uma vez que a quantidade de dados seria muito pequena.

3.2.2 Número de Núcleos

O simulador GEM5 possui uma limitação de 255 núcleos por simulação. Sendo assim, ficou definido que, para este trabalho, estes valores variam entre 16, 32 ou 64 núcleos. A forma como estão dispostas depende do tipo de arquitetura que será utilizada pela simulação, bem como o número de memórias *cache* disponíveis.

3.2.3 Capacidade de Memória Cache L2

A variação da capacidade de memória *cache* L2 por núcleo está entre 16 e 64 kB. Porém, este número na maioria dos casos é diferente do número de núcleos. O número de memórias disponíveis e o tamanho de cada memória varia com o a configuração de arquitetura escolhida.

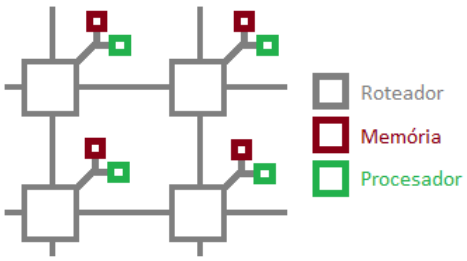
Para a arquitetura em Malha M1, o número de memórias disponíveis é igual ao número de núcleos. Para todas as demais, o número de memórias é igual a um quarto do número de processadores. Por exemplo: uma arquitetura M1 que tenha como configuração 16 núcleos de processamento e 16 kB de capacidade, o número de memórias disponíveis é igual a 16, totalizando 256 kB de memória distribuída. Porém, para uma arquitetura diferente de M1 que tenha como configuração 32 núcleos de processamento e 64 kB de capacidade, temos 8 memórias disponíveis com 256 kB, totalizando os mesmos 2048 kB em memória distribuída, assim como a arquitetura do exemplo anterior.

3.2.4 Tipo de Topologia

O tipo de arquitetura se refere a forma com as quais os elementos (processadores, memórias *cache* L2 e interconexões) estão organizados. Para este parâmetro, pode-se haver três tipos diferentes: malha, *torus* e *cluster*. As topologias precisam ser modeladas para serem usadas como parâmetros do simulador.

São cinco arquiteturas do tipo malha, representadas pela letra M (M0 à M4), quatro arquiteturas do tipo *torus* (T1 à T4), representadas pela letra T e uma do tipo *cluster*, representada pela letra C.

Figura 6 – Parte da Topologia Malha 0



Fonte: Elaborado pelo Autor

A uma diferença para os roteadores de uma das arquiteturas utilizadas neste trabalho, que é a M0 e será apresentada junto com as outras em outra seção deste trabalho. Seus roteadores apresentam dois encaixes, para uma memória e um processador. A Figura 6 mostra uma região de uma malha desta arquitetura.

3.2.5 Número de Clusters

Com exceção das arquiteturas M0, onde cada roteador possui uma memória e um processador, em todas as outras arquiteturas, são intitulados *clusters* os roteadores que possuem memórias *cache* L2.

Sendo assim, a M0 possui um número de *clusters* igual ao número de núcleos, tendo tamanhos de memória *cache* reduzido. Nas demais, o número de *clusters* é igual a um quarto do número de núcleos, tendo tamanhos de memória *cache* maiores.

3.3 Método de Avaliação dos Resultados Por Tipos de Topologia e Aplicações

Iniciando a primeira parte de três dos resultados, têm-se os resultados das análises por tipos de arquitetura, seja ela malha, *torus* ou *cluster*, e por aplicação. Esta parte é destinada a dizer quais aplicações têm melhores resultados dentro dos tipos de arquitetura, nos resultados por arquitetura, e quais tipos de arquitetura tem melhores resultados por aplicação. O intuito destes resultados é dizer quais os impactos das escolhas independente de outros fatores.

Para tal, são avaliados quatro parâmetros. São eles: desempenho (tempo de execução, em segundos), gasto energético (consumo de potência, em *watts*), taxa de *cache miss* (em porcentagem) e acessos a memória por núcleo (em número de acessos).

Cada um desses parâmetros foi dividido em faixas de valores de tamanhos iguais, afim de se classificar os resultados. As primeiras faixas de valores caracterizam números baixos enquanto as últimas representam números altos. Por exemplo, em uma arquitetura que tenha um desempenho presente na primeira faixa de valores para desempenho e um gasto energético na última faixa de valores para gasto energético implica que esta arquitetura tem um ótimo desempenho e um péssimo gasto energético.

As faixas de valores são geradas de acordo com os valores extremos, ou seja, o menor e o maior valor de cada parâmetro. Sendo assim, a primeira faixa vai do menor valor ao primeiro limiar, a segunda faixa vai do primeiro limiar ao segundo limiar, a terceira faixa vai do segundo ao terceiro e assim sucessivamente até a última faixa de valores, que vai do último limiar até o maior valor.

Como um exemplo prático, foi retirado dos resultados os valores de tempo máximo e mínimo da aplicação LU, aproximadamente 3,2 segundos e 0,9 segundos, respectivamente. Assim, os cinco intervalos foram:

- Primeira faixa: de 0,9 a 1,36 segundos. Todos os valores que obtiverem esses valores serão considerados na primeira faixa.

- Segunda faixa: de 1,36 a 1,82 segundos. Todos os valores que obtiverem esses valores serão considerados na segunda faixa.
- Terceira faixa: de 1,82 a 2,28 segundos. Todos os valores que obtiverem esses valores serão considerados na terceira faixa.
- Quarta faixa: de 2,28 a 2,74 segundos. Todos os valores que obtiverem esses valores serão considerados na quarta faixa.
- Última faixa: de 2,74 a 3,2 segundos. Todos os valores que obtiverem esses valores serão considerados na última faixa.

Porém, ao classificar os resultados apenas por arquitetura ou por aplicação, perde-se a referência em dizer qual a arquitetura ou aplicação melhor classificada. Assim, foi preciso criar uma nova seção para os resultados gerais.

3.4 Método de avaliação baseado em AFC

O objetivo principal deste trabalho é apresentar a AFC como técnica de análise de resultados de arquiteturas *many-core*. Sendo assim, foi feita uma análise de outra forma a fim de se comparar os resultados achados por esta análise e pela AFC.

As regras mais fortes, obviamente, são as que possuem 100% de confiança e um alto nível de suporte. Foram estas analisadas primeiro a fim de ser gerar conceitos. As outras são melhores detalhadas pois possuem exceções que impossibilitam a criação de conceitos, mas se pode gerar regras de comportamento das cargas com o nível de suporte considerado.

É válido dizer que os conceitos gerados pela AFC não são absolutas. É preciso mais estudos, variáveis, arquiteturas e outras formas de comprovação nas implicações que são geradas. Porém, é um bom indicativo e pode muito bem caracterizar um *benchmark*, que é o que foi feito com o CAP Bench.

Foram gerados dois tipos de conhecimento: as regras e conceitos por aplicação e as regras e conceitos de todas as aplicações em conjunto. A primeira tem o intuito de encontrar quais as características de cada aplicação com o intuito de se descobrir seu comportamento dado as arquiteturas.

As regras e conceitos sobre todas as aplicações visam a classificação de cada aplicação dentro do conjunto de aplicações. O objetivo é encontrar comportamentos entre as aplicações que possam destacar umas das outras.

4 RESULTADOS

Nesta seção, cada aplicação é analisada pela avaliação convencional e em seguida pela AFC. Sendo assim, cada aplicação é avaliada pelas duas formas e, em seguida, é feito uma comparação entre os resultados.

As regras e conceitos gerados pela AFC são utilizados pra se determinar, quando possível, o comportamento de cada arquitetura perante cada aplicação. Os conceitos mostram o que acontece com a arquitetura independente dos dados de entrada modificados enquanto as regras mostram um possível comportamento predominante.

Para a análise de resultados, sessenta (60) arquiteturas diferentes executaram todas as sete (7) aplicações citadas anteriormente, resultando em um número de 420 execuções ao longo de todo trabalho.

Todas as análises foram feitas após a discretização dos dados. Utilizar os dados diretamente obtidos pelo simulador é incompatível com a utilização da AFC pois é preciso classificar os valores utilizados a fim de agrupá-los. Assim, a análise convencional foi feita também em cima dos dados discretizados.

Pela avaliação convencional, o consumo de potência não apresenta nenhuma alteração em relação as arquiteturas a não ser pela alteração do número de núcleos dentro da rede. Alterar o número de núcleos não é simplesmente aumentar ou reduzir os núcleos, mas também o número de elementos, como memórias *cache*, roteadores e ligações entre os elementos.

Sendo assim, há uma seção separada para tratar o consumo de potência, explicando os resultados para todas as aplicações.

Ao final de cada subseção é feita uma comparação geral entre os dois. O intuito da comparação é evidenciar o que a AFC foi capaz de classificar sobre o que foi descoberto. Vale ressaltar que não foram adicionados retículados de cada conjunto de regras gerados uma vez que seus tamanhos não contribuem para a avaliação.

Tabela 4 – Tamanho de cargas de trabalho *default* por aplicação

Aplicação	Tamanho da Carga
FAST	4028 x 4028
FN	$8x10^6 + 1$ até $8x10^6 + 2^{12}$
GF	8192 x 8192 (i) 11 x 11 (m)
IS	2^{35} inteiros
KM	$2^{14}R^{16}$ pontos 512 centroides
LU	matriz 1536 x 1536
TSP	17 cidades

4.1 Avaliação por Aplicação

A título de simplificar a avaliação, as aplicações são representadas por suas respectivas siglas. As arquiteturas são representadas por uma letra representando a topologia e um número representando a posição das memórias *cache* L2 dentro da rede de roteadores, sendo:

- M0 - arquitetura em Malha original (onde há um processador e uma memória *cache* L2 por roteador.
- M1 à M4 - arquitetura em malha diferenciados pelas posições das memórias *cache* L2.
- T1 à T4 - arquitetura em *torus* diferenciados pelas posições das memórias *cache* L2.
- CL - arquitetura em *cluster*, sendo a única arquitetura organizada neste tipo de topologia.

As cargas de trabalho *default* foram escolhidas para avaliar os resultados de cada aplicação. A Tabela 4 mostram os tamanhos da carga *default* para cada aplicação.

As regras geradas não tem um padrão em específico. Algumas aplicações obtiveram regras mais claras que as outras, demonstrando assim um comportamento mais claro e definido que as demais. As aplicações que apresentaram menos regras e/ou com um grau de confiança menor evidenciam o fato de não terem um comportamento bem definido.

Foram chamados "dados de acontecimento"as características de entrada das regra/-conceitos. Foram chamados "dados de objetivo"as características de saída das regras/conceitos. Suporte é a quantidade de ocorrências dentro das simulações.

As classificações sobre os resultados seguem o padrão mostrado na Tabela 5.

Tabela 5 – Siglas de Classificações

Sigla	Significado
TO	Tempo Ótimo
TB	Tempo Bom
TM	Tempo Médio
TR	Tempo Ruim
TP	Tempo Péssimo
BC	Baixo Consumo de Potência
MC	Médio Consumo de Potência
AC	Alto Consumo de Potência
AMO	Acesso à Memória Ótimo
AMB	Acesso à Memória Bom
AMM	Acesso à Memória Médio
AMR	Acesso à Memória Ruim
AMP	Acesso à Memória Péssimo
CMO	*Cache Miss* Ótimo
CMB	*Cache Miss* Bom
CMM	*Cache Miss* Médio
CMR	*Cache Miss* Ruim
CMP	*Cache Miss* Péssimo

Fonte: Elaborado pelo Autor

4.1.1 FAST

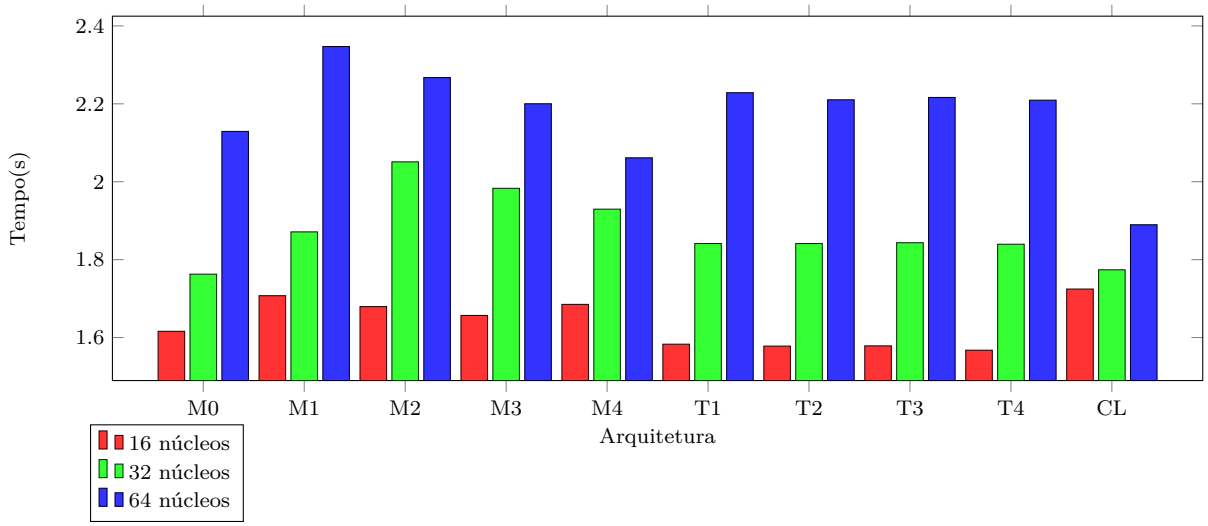
Em geral, a mudança na organização das arquiteturas não é de grande influência no tempo de execução, mas sim a quantidade de núcleos. Em todas as topologias, utilizar uma quantidade maior de núcleos resultou na queda de desempenho, demandando mais tempo para ser executado. A Figura 7 mostra os gráficos de tempo da aplicação FAST, separados por quantidade de núcleos por arquitetura (as barras vermelhas representam arquiteturas de 16 núcleos, as barras verdes representam arquiteturas de 32 núcleos e as barras azuis as de 64 núcleos).

Arquiteturas com menos núcleos, ou seja, menos componentes, apresentam melhores resultados que as demais. Pode-se perceber isso com a Figura 7. As melhores arquiteturas nesse quesito são as organizadas em *torus*, todas com 16 núcleos. Todavia, arquiteturas com maior número de componentes apresentam resultados piores. A arquitetura que apresenta o pior resultado é a M0 com 64 núcleos.

Pela natureza da aplicação, percebe-se que o aumento do número de núcleos, e consequentemente no tamanho da arquitetura, impacta na perda de desempenho, ou seja, aumenta o tempo necessário para execução do algoritmo. Isso ocorre devido ao fato da dependência de informação entre os dados do algoritmo.

A Figura 7 ainda apresenta o impacto que o número de núcleos tem no desempenho

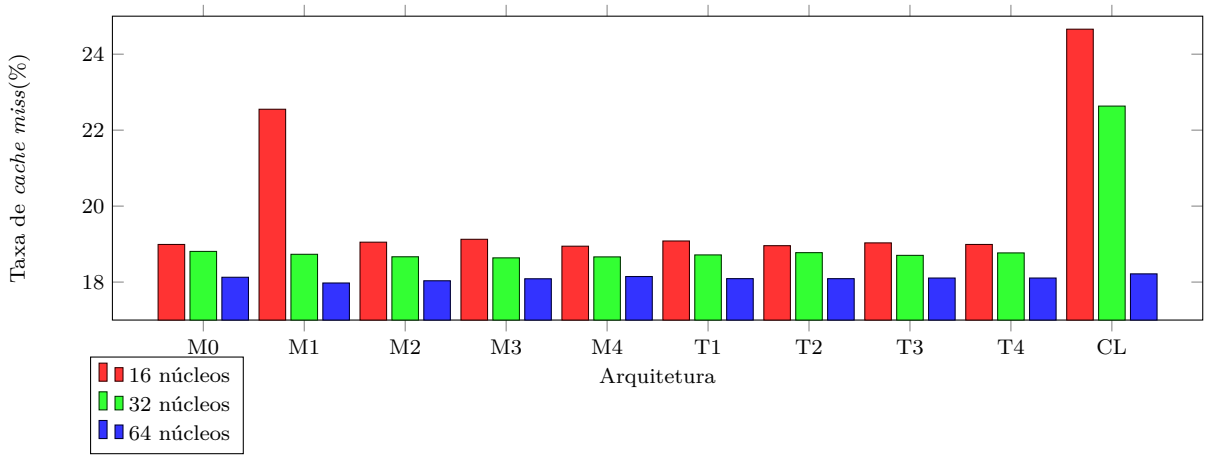
Figura 7 – Gráfico de Tempos da Aplicação FAST



Fonte: Elaborado pelo Autor

do sistema. Enquanto as arquiteturas organizadas pela topologia *cluster* apresentam um impacto menor na variação do número de componentes na rede. As demais apresentam um impacto maior na variação do número de componentes na rede. A Figura 8 mostra o gráfico de taxa de *cache miss* por arquitetura e por núcleo, também diferenciando por número de núcleos.

Figura 8 – Gráfico de taxas de *cache miss* da Aplicação FAST



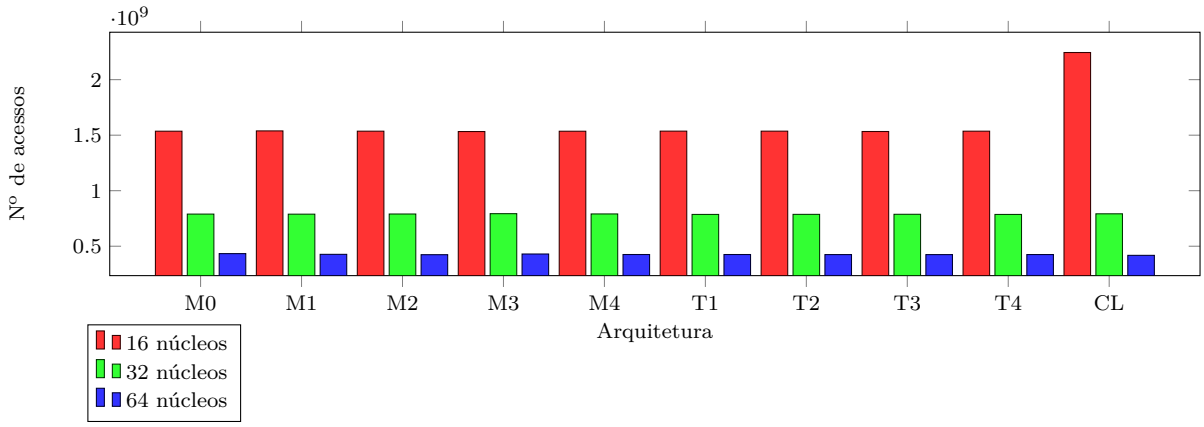
Fonte: Elaborado pelo Autor

Percebe-se que a arquitetura com maior taxa de *cache miss* é a arquitetura organizada em *cluster*. Uma hipótese para este comportamento, decorrente da topologia e conectividade da rede, se deve à alteração do momento de requisição de dados, mudando os endereços mapeados em *cache* e ocasionando maior taxa de *miss*.

Arquiteturas com 64 núcleos apresentam uma taxa de *cache miss* menor do que

as demais. Havendo mais memórias espalhadas pela arquitetura, a taxa diminui, como mostra a Figura 8. A Figura 9 apresenta o gráfico de acessos a memória da aplicação FAST por arquitetura e por número de núcleos.

Figura 9 – Média de Acessos a Memória da Aplicação FAST por núcleo



Fonte: Elaborado pelo Autor

Como um todo, todas as arquiteturas apresentam os menores números de acesso a memória, com exceção da *cluster* que possui 16 núcleos. Se tratando de número de núcleos, quanto mais núcleos menor será o acesso médio à memória.

Pela Figura 9, percebe-se que o acesso médio de memória em cada núcleo diminui de acordo com a quantidade de núcleos. Porém, não há uma queda de acesso completamente proporcional ao número de núcleos. Isto ocorre devido a natureza da aplicação. Cada região utilizada para o cálculo de regiões necessita de um pixel e de seus vizinhos. Sendo assim, as regiões destinadas a cada paralelização do algoritmo compartilham de alguns pixels. Sendo assim, há um aumento no número de acessos a memória *cache*.

A Tabela 6 mostra as regras e conceitos selecionados para a aplicação FAST dentre as 60 simulações executadas para esta aplicação.

Tabela 6 – Regras/Conceitos da Aplicação FAST

Tipo	Confiança(suporte)	Dados de Acontecimento	Dados de Objetivo
Conceito	100%(20)	16 Processadores	TO / BC
Conceito	100%(20)	32 Processadores	AMO
Conceito	100%(20)	64 Processadores	CMR / AMO
Conceito	100%(15)	TB	CMR / AMO
Regra	88%(60)	-	CMR
Regra	65%(60)	-	BC
Regra	90%(30)	64 kB	CMR
Regra	88%(24)	TO	BC
Regra	70%(20)	64 processadores / CMR / AMO	AC

Fonte: Elaborado pelo Autor

Os conceitos mostram que o tempo de execução é ótimo, ou seja, menor que os demais, quando se tem 16 processadores e o gasto energético é baixo. Quando executado em arquiteturas com 32 processadores, o acesso a memória é ótimo e quando é executado em arquiteturas com 32 processadores, o *cache miss* é maior porém o acesso a memória continua ótimo.

Sobre as regras, 88% dos resultados mostraram um *cache miss* ruim e 65% dos resultados apresentaram um baixo consumo de energia. Quando o tempo de execução é ótimo, a porcentagem de baixo consumo aumenta para 88%. Em 70% das simulações com 64 processadores que apresentaram um *cache miss* ruim e um acesso a memória ótimo resultaram em um alto consumo de potência.

Os resultados indicam que arquiteturas com 16 processadores possuem um baixo consumo de potência independente de outros fatores. Este fator, como dito anteriormente, mostra a grande influência que o número de elementos tem dentro de uma arquitetura quando o assunto é consumo de potência.

As regras também apontam indícios de que, independente das configurações utilizadas, os resultados apresentaram um baixo consumo (65% dos resultados) e/ou um *cache miss* ruim (88%). Arquiteturas com 64 kB de memória terão um *cache miss* ruim (90%) e execuções com tempo ótimo terão um baixo consumo de energia (88%).

A aplicação FAST mostrou na avaliação convencional que arquiteturas com um número menor de processadores tem um melhor desempenho em relação as outras. Mostrou um acesso a memória estável e também um consumo de potência previsível.

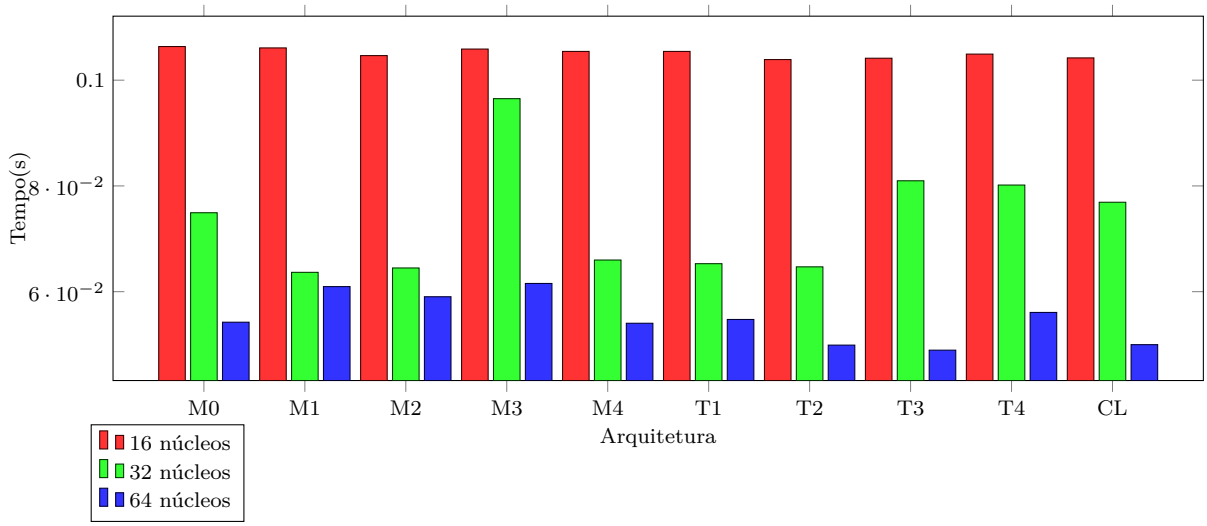
Pela AFC, as regras e conceitos a cerca dessas aplicações mostraram que 65% das execuções ou 88 % das execuções que possuem um tempo de execução ótimo tiveram um baixo consumo. Arquiteturas com 64 processadores também mostram uma taxa de *cache miss* ruim, porém um ótimo acesso a memória.

4.1.2 FN

A aplicação FN foi a aplicação de menor média de tempo de execução dentre as demais aplicações. Novamente influenciada pelo número de *clusters* assim como a aplicação anterior com a diferença de que quanto maior o número de *clusters*, maior o desempenho apresentado pela arquitetura. A Figura 10 mostra o desempenho por arquitetura e por número de núcleos da aplicação FN.

O comportamento aleatório das arquiteturas mostra que não há um padrão pela topologia das arquiteturas. Tanto a topologia *torus* quanto a topologia em malha apresentam valores variados de desempenho. Essa indefinição de comportamento por topologia é evidenciada pela arquitetura M3 em relação as demais arquiteturas organizadas em malha

Figura 10 – Gráfico de Tempos da Aplicação FN

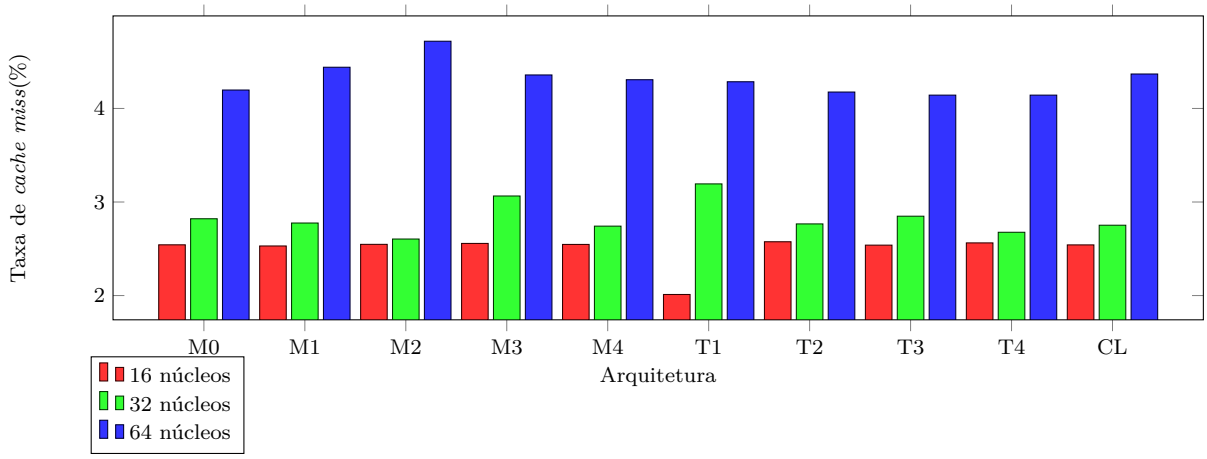


Fonte: Elaborado pelo Autor

e as arquiteturas T1 e T2 em relação as arquiteturas T3 e T4.

A Figura 10 também mostra o desempenho em relação ao número de processadores. Pela natureza da aplicação, é possível perceber que quanto maior a quantidade de núcleos, melhor o desempenho, ou seja, menos tempo será necessário para se executar a aplicação FN. As arquiteturas com 64 núcleos, representadas pelas barras azuis do gráfico, apresentam os baixos valores de tempo em relação as demais. A Figura 11 mostra os gráficos de taxa de *cache miss* por arquitetura e por número de núcleos.

Figura 11 – Gráfico de taxas de *cache miss* da Aplicação FN



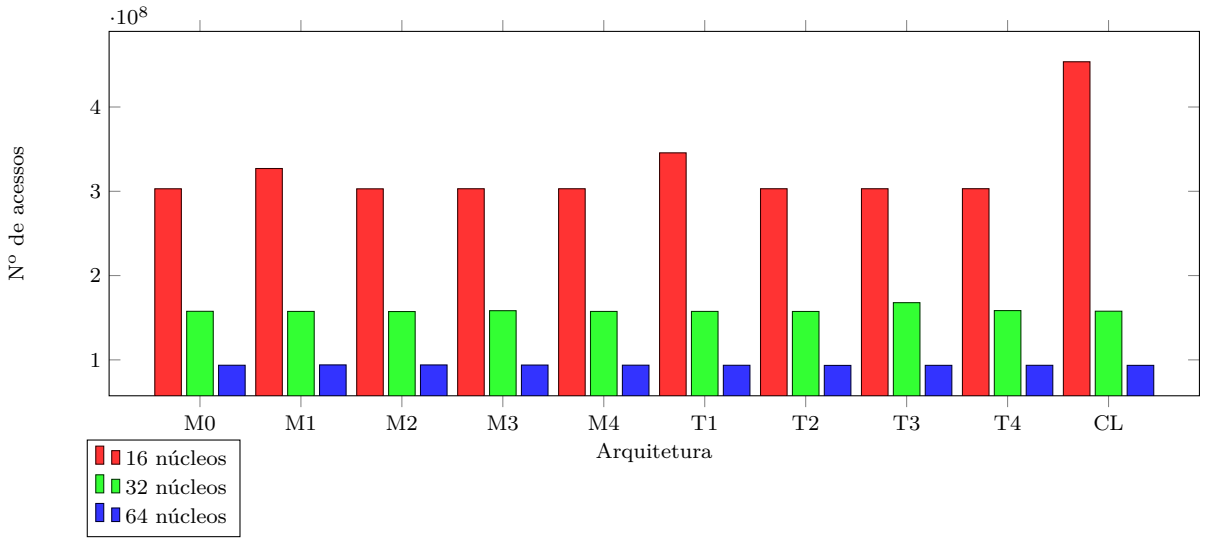
Fonte: Elaborado pelo Autor

A taxa de *cache miss* da aplicação FN mostrou-se dependente da posição das memória dentro da rede-em-chip, porém nada muito significativo. Percebe-se que metade das arquiteturas que são correspondentes (M1 com T1 e M3 com T3) apresentam com-

portamentos semelhantes. As ligações que se dão nas outras (M2, M4, T2 e M4) fazem com que apresentem comportamentos diferentes.

A Figura 12 apresenta o gráfico de acesso médio a memória da aplicação FN.

Figura 12 – Média de Acessos a Memória da Aplicação FN por núcleo



Fonte: Elaborado pelo Autor

Através da Figura 12, pode-se ver novamente um comportamento semelhante entre as arquiteturas correspondentes. Quanto mais ligações, melhor o acesso a memória principal. As arquiteturas organizadas em *cluster* tem menos ligações do que todas as outras, mostrando um acesso a memória maior que os demais.

O segundo critério é a posição das memórias *cache* dentro da rede, apresentando números mais baixos para arquiteturas 3 (M3 e T3) e números maiores para arquiteturas 1 (M1 e T1).

Assim como a aplicação anterior, FN mostrou um comportamento de acessos a memória médio menor dependendo da quantidade de núcleos. Porém, o desempenho das tarefas paralelizadas por esta aplicação não dependem de acesso a memória ou da organização da topologia, como podemos perceber nas três Figuras apresentadas.

A Tabela 7 mostra as regras e conceitos selecionados para a aplicação FN dentre as 60 simulações executadas para esta aplicação.

Tabela 7 – Regras/Conceitos da Aplicação FN

Tipo	Confiança(suporte)	Dados de Acontecimento	Dados de Objetivo
Conceito	100%(22)	TP	BC
Conceito	100%(20)	16 Processadores	TP / BC
Conceito	100%(20)	32 Processadores	AMO
Conceito	100%(20)	64 Processadores	AC / CMP / AMO
Conceito	100%(13)	32 Processaodres / TB / AMO	BC
Regra	67%(60)	-	AMO
Regra	63%(60)	-	BC
Regra	94%(32)	CMM	BC
Regra	80%(10)	64 Processadores / 16 kB / AC / CMP / AMO	TO
Regra	80%(10)	64 Processadores / 64 kB / AC / CMP / AMO	TO

Fonte: Elaborado pelo Autor

A aplicação FN se mostrou menos previsível que a aplicação anterior. Porém, alguns conceitos mostraram comportamentos da aplicação que chamam a atenção. Com 100% confiança e com um suporte de 22 ocorrências sobre 60 execuções, os resultados que mostram um tempo de execução péssimo também mostram um baixo consumo de energia.

Já era esperado que quanto menor o número de processadores, menor o consumo, como o segundo conceito da Tabela 7 mostra. O quarto conceito também mostra que quanto maior o número de processadores, e consequentemente maior o número de elementos de toda arquitetura, maior o consumo de potência.

Não houve nenhum conceito que mostrasse um padrão a ser seguido para se ter um tempo de execução ótimo, porém duas regras mostrou este indício. As duas últimas regras apresentadas na Tabela 7 mostram um conjunto de fatores que levam ao melhor tempo de execução.

Sendo assim, pode-se dizer que para alcançar um bom desempenho com essa arquitetura, é preciso alcançar uma série de fatores. Dadas as regras que abordam este valor de desempenho, tem-se que é preciso um alto consumo de potência, uma taxa de *cache miss* péssima e um acesso a memória ótimo/mínimo.

Um bom ponto a se observar também é sobre o que o quinto conceito tem a dizer. Ele afirma que 100% das vezes em que uma arquitetura de 32 processadores obteve um tempo bom e um acesso a memória ótimo, o consumo de energia foi baixo.

A aplicação FN apresentou padrões aleatórios de comportamento, não sendo possível identificar certos aspectos a não ser uma predisposição que as arquiteturas organizadas em malha tem para executar a aplicação em um tempo baixo. Esta avaliação convencional também mostrou que quanto mais processadores, melhor para o desempenho da aplicação

Pela AFC, descobriu-se que tempos péssimos de execução desta arquitetura resultam em um baixo consumo de potência. Indícios apontam que taxas de *cache miss* médias resultam em um baixo consumo de potência também e o tempo ótimo foi alcançado por arquiteturas com 64 processadores, taxas de *cache miss* péssimos e acessos a memória ótimo.

4.1.3 GF

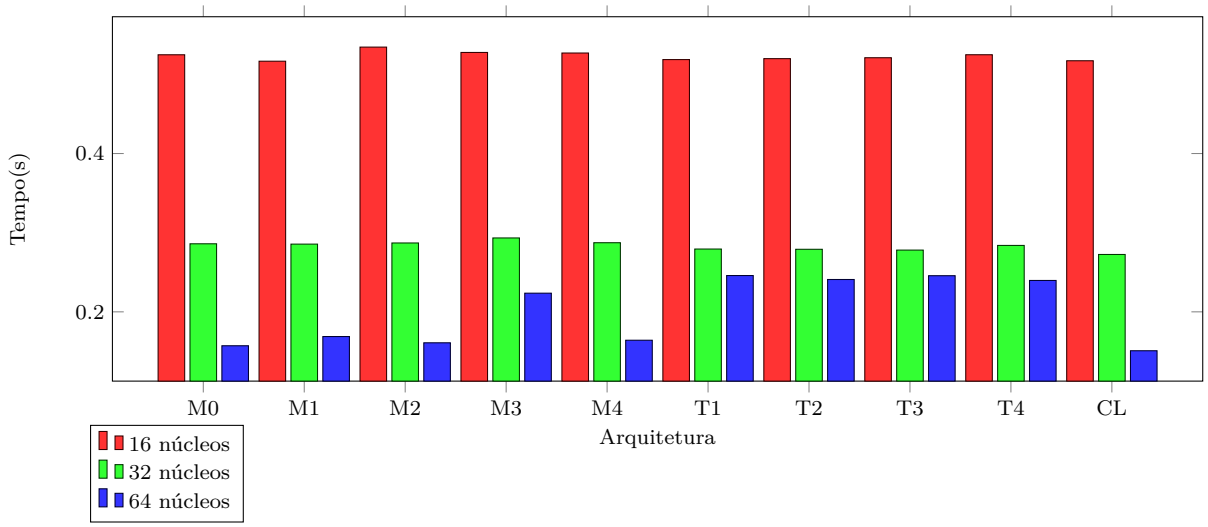
A aplicação GF é a que apresentou a maior estabilidade de tempo de execução e taxa de *cache miss* dentre todas as outras aplicações, mantendo uma média de execução de 0,28 s e 2,65% de *cache miss*.

Pode-se dizer que as arquiteturas que tem o menor caminho médio de distância entre os núcleos obtiveram resultados ligeiramente melhores. A topologia *cluster* teve o menor resultado médio (0,27 segundos) enquanto a topologia em malha obteve o maior resultado médio (0,29 segundos).

A topologia *cluster* possui o menor caminho médio. A organização da topologia coloca uma memória *cache* proporcional a quatro núcleos, ficando a dois roteadores de distância da memória mais próxima e a quatro da mais distante. Pelo fato da aplicação precisar utilizar memória para armazenar os novos valores do filtro, ter mais memória e a uma distância menor aumento

A Figura 13 mostra os tempos de execução por arquitetura e número de núcleos.

Figura 13 – Gráfico de Tempos da Aplicação GF

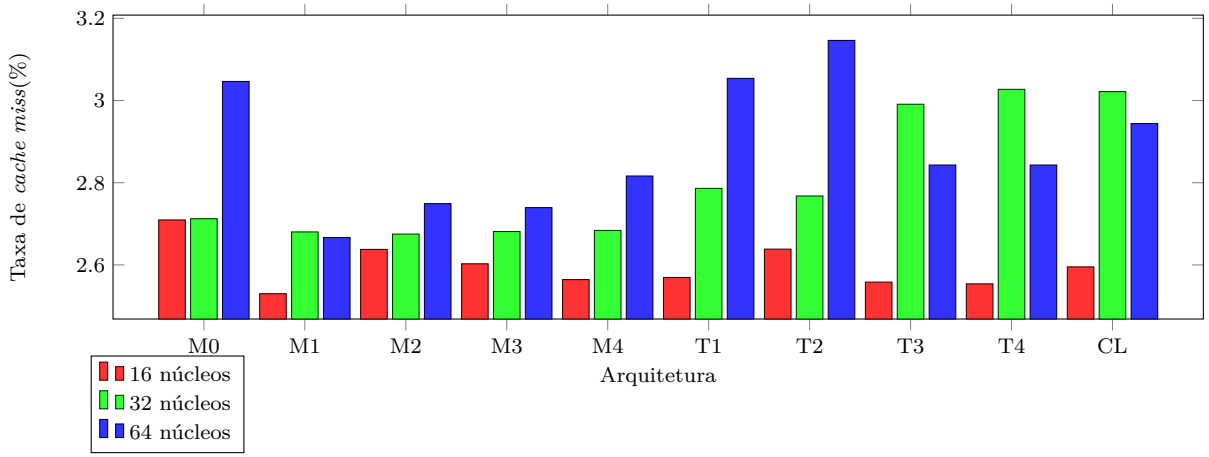


Fonte: Elaborado pelo Autor

Praticamente não faz diferença como está organizada a arquitetura dentre as topologias propostas. Porém, pelo número de processadores é diferente. A Figura 13 também

apresenta o desempenho médio das arquiteturas por número de processadores. Com a Figura, é possível ver que o aumento de processadores melhoram o desempenho da arquitetura.

Figura 14 – Gráfico de taxas de *cache miss* da Aplicação GF



Fonte: Elaborado pelo Autor

A Figura 14 mostra os valores médios de taxa de *cache miss* por arquitetura e por quantidade de núcleos da aplicação GF. Pode-se perceber que as arquiteturas que possuem mais memórias *cache* espalhadas ao longo da rede aumenta a taxa de *cache miss*, como mostra a Figura 14.

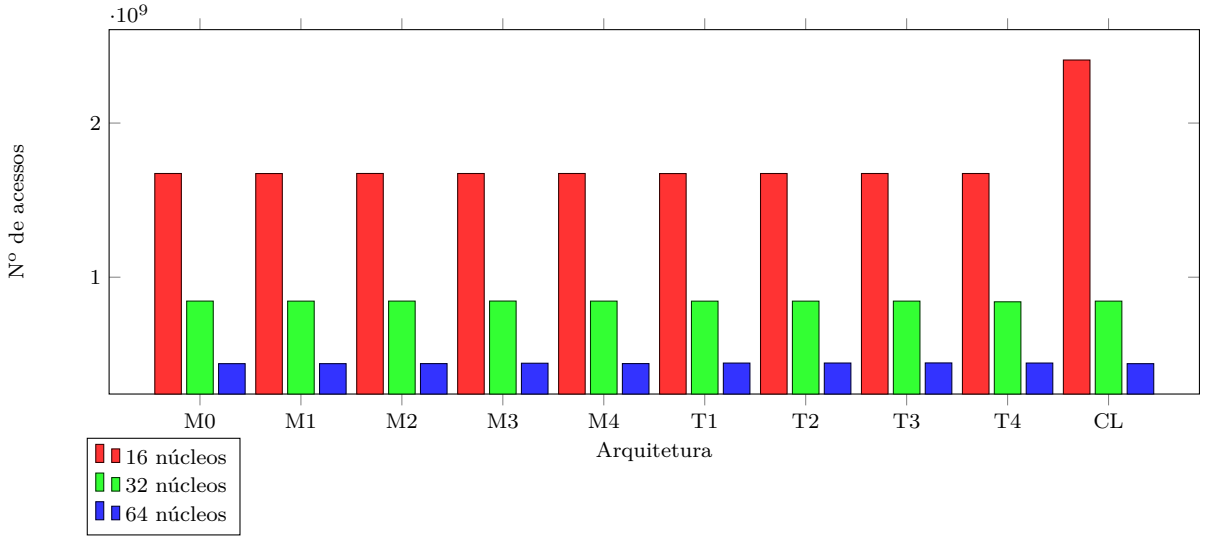
As iterações da aplicação GF tem alta dependência entre si. Isso quer dizer que o resultado da iteração anterior tem muita influência sobre a iteração atual. Isso faz com que a quantidade de *cache miss* aumente em arquiteturas com menos caminhos de comunicação entre o núcleo e a memória do algoritmo, evidenciado na Figura 14.

A Figura 15 apresenta os gráficos de acesso a memória da aplicação GF. Praticamente todas as arquiteturas apresentaram a mesma média de acessos a memória, com exceção feita as arquiteturas organizadas *cluster*. Porém, quando maior a quantidade de elementos (representado pelo aumento da quantidade de núcleos) da arquitetura, menor vai será o número de acesso médio por núcleo.

A aplicação GF tem comportamento similar a aplicação FAST. Ambas tem regiões de interesse a serem processadas a base de pixels. A diferença está no fato da aplicação FAST ser utilizada para detecção. Ainda assim, as regiões extremas de pixels entre um cluster e outro compartilham dos mesmo pixels, fazendo com que o acesso a memória média por núcleo seja menor com arquiteturas com mais núcleos, porém há um aumento geral no acesso a memória.

A Tabela 8 mostra as regras e conceitos selecionados para a aplicação GF dentre as 60 simulações executadas para esta aplicação.

Figura 15 – Médio de Acessos a Memória da Aplicação GF por núcleo



Fonte: Elaborado pelo Autor

Tabela 8 – Regras/Conceitos da Aplicação GF

Tipo	Confiança(suporte)	Dados de Acontecimento	Dados de Objetivo
Conceito	100%(30)	64 kB	CMR
Conceito	100%(20)	16 Processadores	TP / BC
Conceito	100%(20)	32 Processadores	TB / AMO
Conceito	100%(20)	64 Processadores	AMO
Regra	83%(60)	-	CMR
Regra	67%(60)	-	AMO
Regra	70%(50)	CMR	BC
Regra	70%(40)	AMO	TB
Regra	67%(30)	16 kB	AMO
Regra	67%(30)	16 kB	CMR

Fonte: Elaborado pelo Autor

A aplicação GF apresentou menos conceitos que as aplicações anteriores, porém, conceitos com suportes maiores, ou seja, suas regras tem maiores incidências do que as regras das aplicações anteriores até o momento mostraram. Outra peculiaridade desta aplicação em relação as demais está no fato do aumento no número de processadores não influenciar tanto no consumo de potência quanto as demais aplicações.

A Tabela 8 apresenta um conceito interessante. Com 100% das execuções em máquinas com 64 kB de memória *cache* L2 por núcleo, obteve-se uma taxa de *cache miss* ruim. A terceira regra da Tabela mostra que 70% das execuções que obtiveram um *cache miss* ruim também obtiveram um baixo consumo de potência.

Sendo assim, pode-se dizer que o conceito e a regra se complementam para um baixo consumo de potência. O mesmo ocorre com o quarto conceito que afirma que

execuções com 64 processadores obtiveram um acesso a memória ótimo e a quarta regra que afirma que 70% das execuções que obtiveram um tempo de execução bom.

As regras e conceitos então apontam um forte indício que arquiteturas com mais processadores e mais memória, para esta aplicação, terão um consumo de potência baixo e um bom desempenho.

O consumo de potência para arquiteturas com 16 processadores têm um consumo de potência mais baixo que as demais, como as outras aplicações também mostram. Porém, não há uma influência tão grande no consumo de potência o aumento do número de processadores. Isso fica evidente com a falta de regras fortes e conceitos gerados pelo FCA.

Pela avaliação convencional, a aplicação GF obteve comportamentos parecidos na maioria de suas execuções, mantendo um tempo de execução quase igual para todas as médias de tempo analisados por arquitetura. Em relação ao número de processadores, pode-se perceber um aumento do desempenho em conjunto com o aumento do número de processadores.

O AFC mostra que arquiteturas com 64 kB de memória apresentaram uma taxa de *cache miss* ruim. Também mostra que o aumento no número de processadores resulta em um desempenho melhor, além de apontar indícios de que taxas de *cache miss* apresentam um baixo consumo de potência e um baixo número de acessos a memória resulta em um tempo de execução bom.

4.1.4 IS

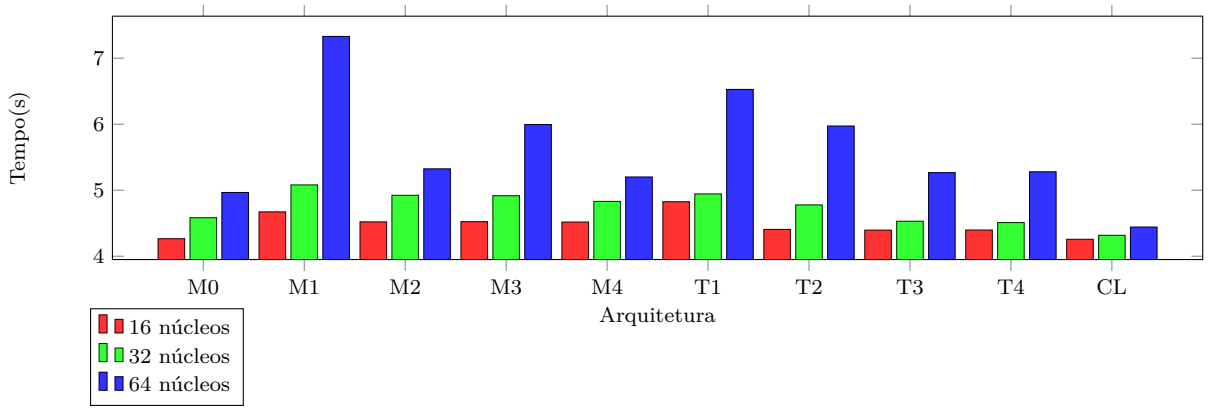
A aplicação IS, ao contrário da aplicação anterior, foi a aplicação com maior tempo de execução dentre todas as outras. Os picos máximos e mínimos de cada arquitetura mantiveram-se estáveis, não havendo muita diferença entre a média e os valores obtidos nas simulações. A Figura 16 mostra os tempos médios de execução por aplicação e núcleo.

A topologia não influenciou consideravelmente e sim a posição das memórias *cache* dentro da rede, como mostra a Figura 16. É possível perceber este comportamento uma vez que os pares de posições de memória similares, ou seja, as arquiteturas em pares (M1, T1) e (M4, T4) apresentam comportamentos semelhantes.

Assim como a aplicação FAST, esta aplicação tem uma grande dependência entre suas iterações, uma vez que é preciso se comparar números para saber qual deles é o maior ou menor. Assim, aumentar a quantidade de núcleos irá aumentar no tempo necessário para a execução do algoritmo.

Se tratando de desempenho por núcleo, como mostrado na Figura 16, temos que

Figura 16 – Gráfico de Tempos da Aplicação IS

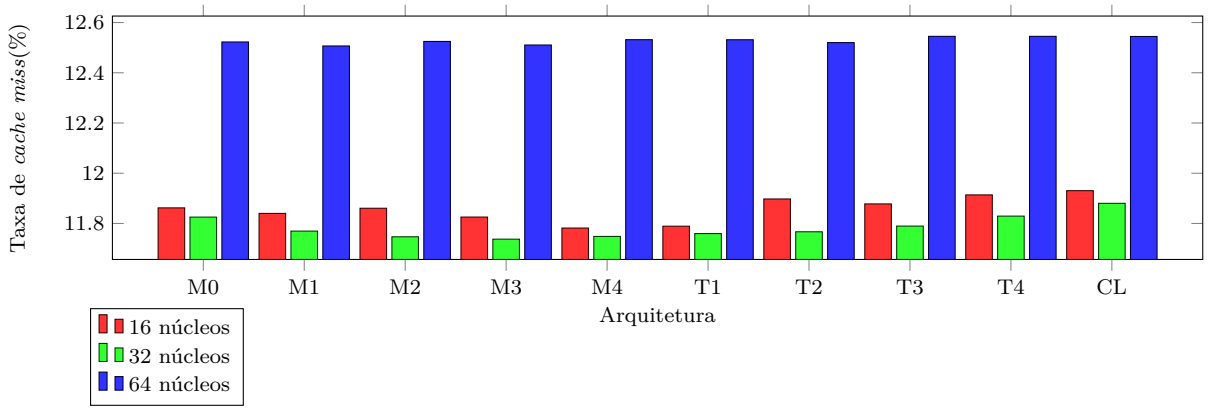


Fonte: Elaborado pelo Autor

arquiteturas com menos núcleos se saem melhor. Por topologias, arquiteturas organizadas em *cluster* possuem os melhores resultados

É perceptível o aumento médio do tempo gasto pela aplicação a medida que se aumenta o número de núcleos da rede. Novamente, a taxa de *cache miss* apresentou-se estável, com valor médio de 12,28%.

Figura 17 – Gráfico de taxas de *cache miss* da Aplicação IS



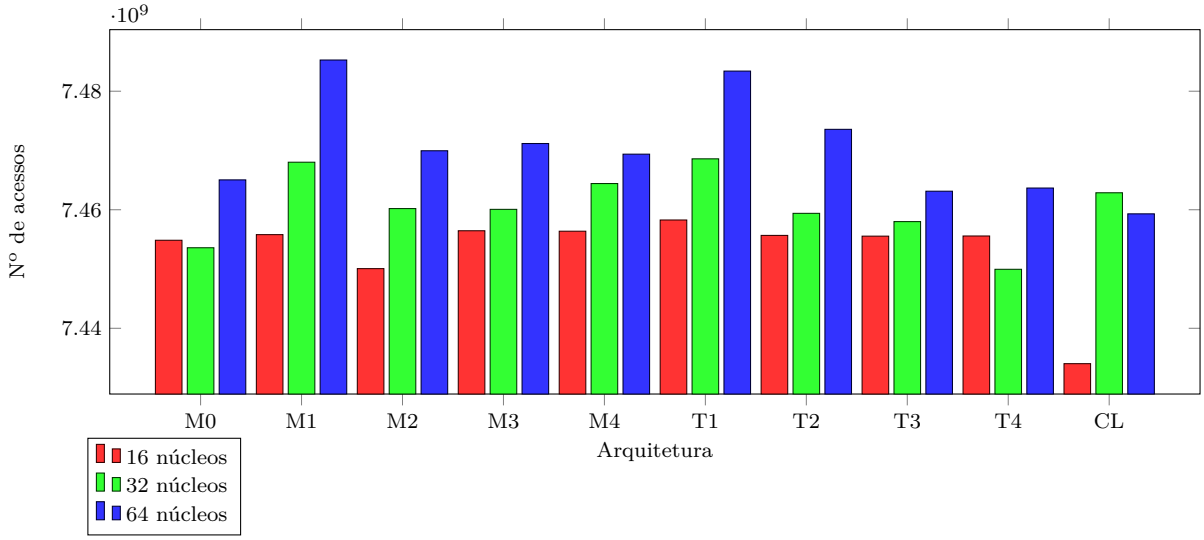
Fonte: Elaborado pelo Autor

A Figura 17 apresenta os gráficos de taxa de *cache miss* por arquitetura e por número de núcleos da aplicação IS. Esta Figura mostra uma taxa de *cache miss* estável no quesito arquitetura. O mesmo se pode afirmar sobre a quantidade de núcleos, havendo uma baixa vairação entre eles. Isso ocorre devido a natureza da aplicação IS, dependendo da ordenação inicial que os números possuem para um bom desempenho.

A Figura 18 apresenta os gráficos de acesso a memória da aplicação IS por arquitetura e por quantidade de núcleos.

Seguindo o exemplo do quesito anterior, o número de acesso a memória se manteve

Figura 18 – Média de Acessos a Memória da Aplicação IS por núcleo



Fonte: Elaborado pelo Autor

estável por arquitetura e pela quantidade de elementos, como evidenciado na Figura 18, exceção feita a arquitetura organizada em *cluster* com 16 núcleos.

Diferente de todas as demais, a aplicação IS tem o maior acesso a memória médio por núcleos dentre todas as outras. Dada a natureza da aplicação, os resultados tem de serem compartilhados entre os núcleos frequentemente, uma vez que é preciso ordenar os números. Dessa forma, o acesso a memória é a forma como os núcleos dividem suas informações para encontrar o resultado correto.

A Tabela 9 mostra as regras e conceitos selecionados para a aplicação IS dentre as 60 simulações executadas para esta aplicação.

Tabela 9 – Regras/Conceitos da Aplicação IS

Tipo	Confiança(suporte)	Dados de Acontecimento	Dados de Objetivo
Conceito	100%(30)	64 kB	CMP / AMP
Conceito	100%(20)	16 Processadores	BC
Conceito	100%(20)	32 Processadores	CMP / AMP
Conceito	100%(20)	64 Processadores	CMP / AMP
Conceito	100%(11)	32 Processadores / TB / CMP / AMP	BC
Conceito	100%(6)	Arquitetura Cluster	TO / CMP
Regra	98%(60)	-	AMP
Regra	63%(50)	-	BC
Regra	60%(20)	64 Processadores / CMP / AMP	AC
Regra	61%(18)	16 kB / BC / AMP	CMR

Fonte: Elaborado pelo Autor

Diferentes das outras aplicações apresentadas até agora, a aplicação IS apresentou

mais regras que resultam em impacto no desempenho, pelo menos em relação a um tempo de execução bom. E tem-se a presença marcante das arquiteturas nas regras geradas para essa aplicação.

O primeiro conceito já apresenta que arquiteturas com 64 kB de memórias *cache* L2 tem uma taxa de *cache miss* péssima, independente de outros valores. Tem-se também uma regra com um grau de confiança e suporte altos dizendo que esta aplicação tem um péssimo/grande acesso a memória principal.

A segunda regra da Tabela 9 também afirma que 63% das execuções resultaram em um baixo consumo de potência. Logo em seguida, a próxima regra afirma que configurações com 64 processadores que possuem uma taxa de *cache miss* péssima e um péssimo acesso a memória resultam em um alto consumo de potência.

O sexto e último conceito chama a atenção para o fato de que todas as simulações executadas em arquiteturas do tipo *cluster* resultaram em um tempo de execução ótimo e uma taxa de *cache miss* péssima. Esta foi a aplicação que mais gerou conceito sobre as arquiteturas que as demais.

O consumo de potência começa a apresentar um comportamento semelhante. Arquiteturas com 16 processadores apresentam um baixo consumo de potência, porém o aumento de processadores não deixa tão claro que as arquiteturas consumiram mais potência.

Pela avaliação convencional, a aplicação IS foi a que mais demandou tempo de execução, variando bastante os picos. A arquitetura organizada em topologia *cluster* obteve os melhores resultados e a posição da memória na rede influenciou bastante seu desempenho.

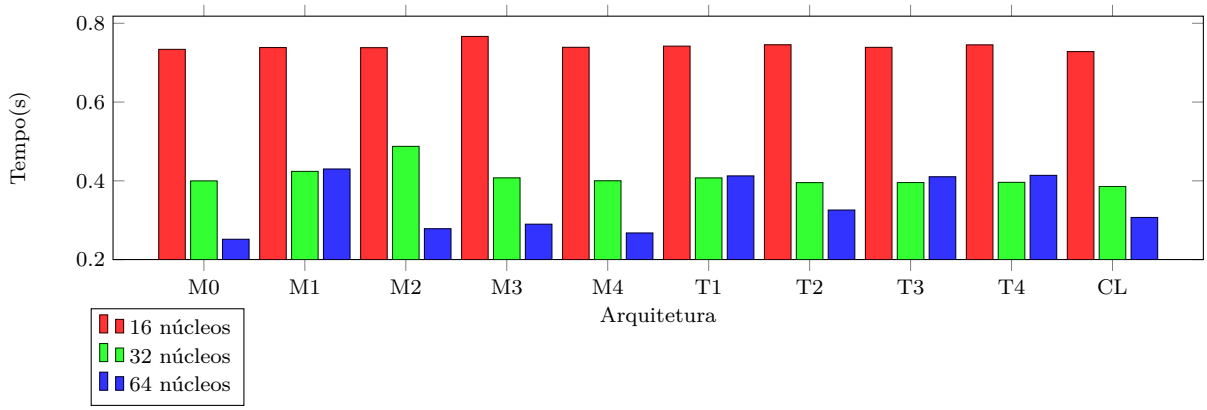
A segunda avaliação, feita utilizando-se a AFC, mostra que a maioria das execuções tem um consumo de potência baixo e também identifica o sucesso obtido pela arquitetura *cluster*.

4.1.5 KM

A aplicação KM não apresenta um comportamento semelhante entre arquiteturas organizadas pela topologia em malha. As outras se mantiveram semelhantes. A Figura 19 mostra os tempos de execução por arquitetura.

As arquiteturas organizadas em *cluster* se saíram melhores que as demais enquanto as arquiteturas M2 mostraram os piores resultados, apesar de não apresentarem valores significativamente diferentes. Porém, a diferença de desempenho por arquitetura não apresenta mudanças significantes.

Figura 19 – Gráfico de Tempos da Aplicação KM



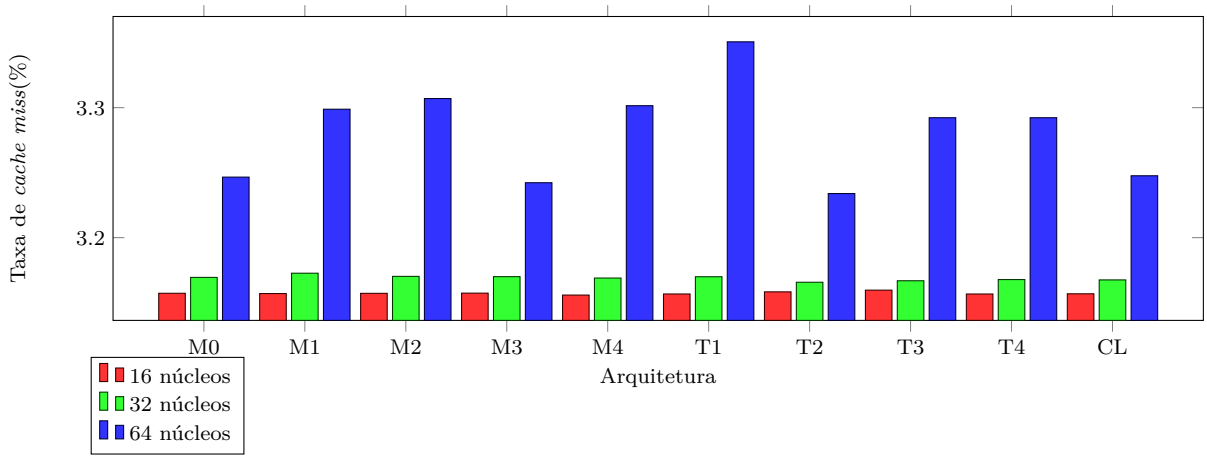
Fonte: Elaborado pelo Autor

A Figura 19 também mostra o desempenho por quantidade de processadores. Com o aumento do número de processadores, ouve o aumento do desempenho, todavia se percebe que o ganho de desempenho entre sistemas com 16 para 32 núcleos é maior que de 32 para 64 núcleos, mostrando que o aumento do número de núcleos poderá não impactar consideravelmente no desempenho em um certo limiar de núcleos.

Curiosamente, algumas arquiteturas apresentaram perda de desempenho quando o número de núcleos variou de 32 para 64 núcleos. São elas: M1, T1, T3 e T4. Essas arquiteturas evidenciam o fato de que aumentar o número de núcleos do sistema pode não ser a melhor alternativa para ganho de desempenho.

Como as simulações foram feitas com cargas de trabalho de mesmo tamanho, a quantidade de núcleos que irá estabilizar o desempenho é fixa. Se aumentar ou diminuir a carga de trabalho, então se aumentaria ou diminuiria, respectivamente, esta quantidade de núcleos necessários para a estabilização.

Figura 20 – Média de taxas de *cache miss* da Aplicação KM por núcleo

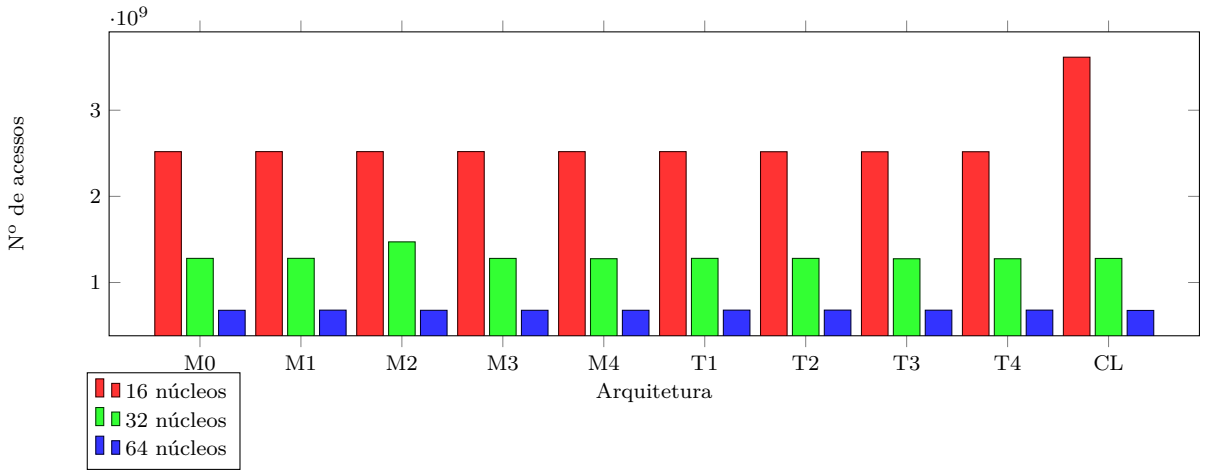


Fonte: Elaborado pelo Autor

A Figura 20 mostra os gráficos de taxa de *cache miss* da aplicação KM por arquitetura e por quantidade de núcleos. A média por arquiteturas é praticamente a mesma em todas as simulações. Há uma pequena alteração pela quantidade de núcleos mas também não apresenta grandes mudanças.

A Figura 21 mostra a quantidade de acessos a memória principal pela aplicação KM por arquitetura e por quantidade de núcleos. Tratando-se de arquitetura, não há diferença significativa. Já por quantidade de núcleos, percebe-se que o número de acessos a memória por núcleo diminui a medida que a a quantidade de núcleos aumenta. Isso ocorre devido a natureza da aplicação. Mantendo o tamanho da carga de trabalho, quanto mais núcleos, menos cálculo por núcleos será necessário.

Figura 21 – Gráfico de Acessos a Memória da Aplicação KM



Fonte: Elaborado pelo Autor

Uma vez que os clusters estão definidos, não é preciso movimentar constantemente as memórias do sistema. Sendo os pontos centrais dentre cada clusters os únicos valores que mudam na execução do algoritmo, o acesso a memória se mantém constante.

A Tabela 10 mostra as regras e conceitos selecionados para a aplicação KM dentre as 60 simulações executadas para esta aplicação.

Tabela 10 – Regras/Conceitos da Aplicação KM

Tipo	Confiança(suporte)	Dados de Acontecimento	Dados de Objetivo
Conceito	100%(39)	AMO	CMP
Conceito	100%(30)	64 kB	BC
Conceito	100%(20)	16 Processadores	TP / BC
Conceito	100%(20)	32 Processadores	CMP
Conceito	100%(20)	64 Processadores	CMP / AMO
Regra	98%(60)	-	CMP
Regra	72%(39)	CMP / AMO	TB
Regra	80%(20)	64 Processadores / CMP / AMO	AC
Regra	56%(16)	64 Processadores / AC / CMP / AMO	TO
Regra	80%(10)	64 Processadores / 64 Kb / CMP / AMO	AC

Fonte: Elaborado pelo Autor

A aplicação KM pode ser caracterizada por algumas regras com valores altos de confiança e suporte, tornando-se até então a aplicação mais previsível. Isso é devido a sua natureza de pouco acesso a memória principal (acesso ótimo) e maior utilização das memórias *cache* (taxa de *cache hit* ótimo).

A Tabela 10 apresenta dois conceitos e uma regra que tem suportes e confiança bem altos. O primeiro conceito mostra que onde há um acesso a memória ótimo/baixo há uma taxa de *cache miss* péssima/alta. O segundo mostra que arquiteturas com memórias *cache* de 64 kB apresentam um baixo consumo de potência.

Em seguida, uma regra mostra que 98% das execuções apontam uma taxa de *cache miss* péssima/alta. Logo em seguida, a segunda regra mostra que essa mesma medida em conjunto com um acesso a memória ótimo resultam em um tempo de execução bom.

Em seguida, aparecem regras mais específicas. O conhecimento gerado em torno de arquiteturas com 64 processadores apresentam um bom número de regras. Como exemplo, a terceira regra mostra que 80% das ocorrências mostram que arquiteturas com 64 processadores, uma taxa de *cache miss* péssima/alta e um acesso ótimo a memória implicam em um consumo de potência alto.

Seguindo o mesmo padrão das anteriores, as arquiteturas com 16 processadores sempre possuem um consumo de potência baixo. Porém, além disso, também mostram um péssimo desempenho em relação as demais.

A aplicação KM pela avaliação convencional apresenta um padrão de execução também bem parecido com as demais, variando de 0,39 à 0,49 segundos. Além disso, o desempenho da aplicação é maior em arquieturas com mais processadores.

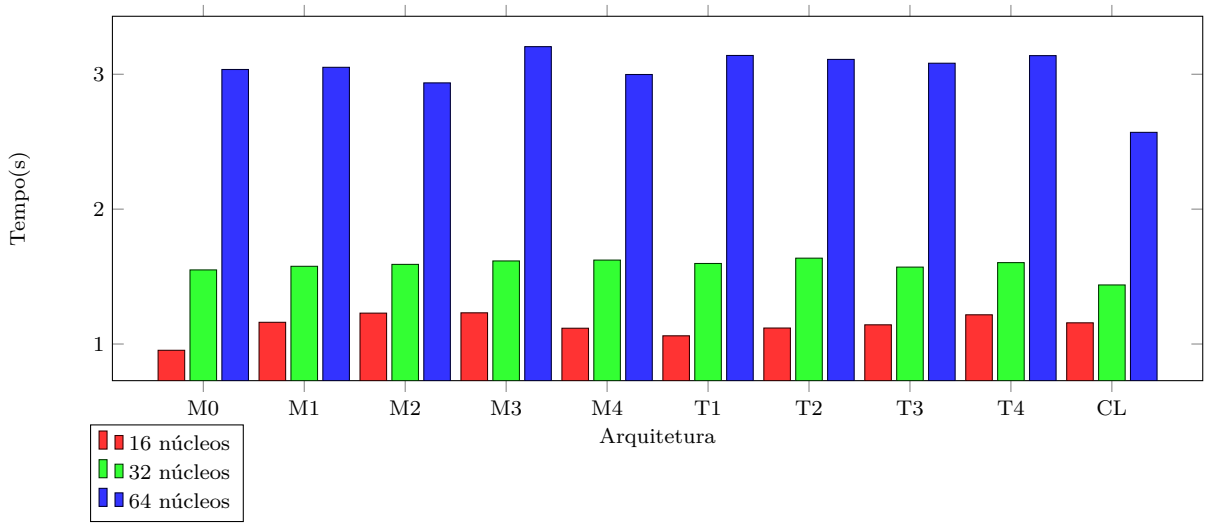
Pela AFC, foi descoberto que arquiteturas com 64 kB de memória apresentam um baixo consumo de potência, a maioria de suas execuções resultam em uma taxa de *cache*

miss péssima e 72% das execuções com taxa de *cache miss* péssima e acesso a memória ótimo tem um tempo de execução bom.

4.1.6 LU

Na aplicação LU, pode-se perceber um certo impacto da topologia no desempenho. A Figura 22 mostra o desempenho médio de cada arquitetura e de cada quantidade de núcleos para a aplicação.

Figura 22 – Gráfico de Tempos da Aplicação LU



Fonte: Elaborado pelo Autor

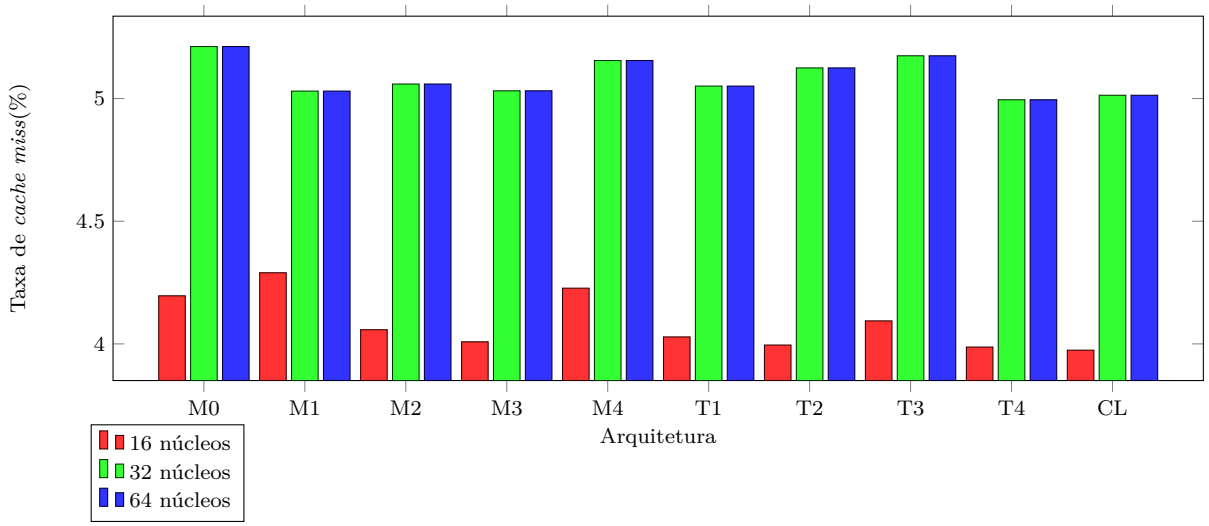
A melhor dentre as três foi a topologia *cluster*, seguida por um desempenho ligeiramente melhor das arquiteturas organizadas em malha e por último as arquiteturas organizadas em *torus*.

Como as aplicações FAST e IS, a aplicação LU também não apresentou melhoras com o aumento do número de processadores. O calculo das matrizes LU possui uma grande dependência entre suas iterações, fazendo com que uma maior fragmentação dos dados causa perda de desempenho.

A Figura 23 apresenta o gráfico de taxas de *cache miss* para a aplicação LU por topologia e quantidade de núcleos. Ao contrário da aplicação anterior, a aplicação LU apresentou um aumento no tempo de execução quando o número de núcleos aumentou. A taxa de *cache miss* se mostra constante em 5,01% e o número de acessos a memória se mostra pouco semelhante entre as simulações para essa aplicação. A Figura 24 mostra esse comportamento dentre cada arquitetura.

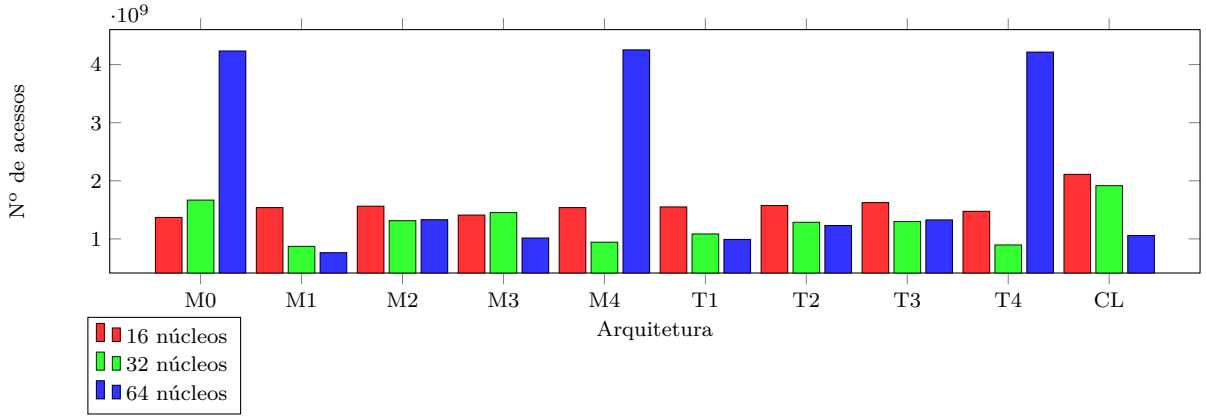
A aplicação LU possui um comportamento incomum dentre as demais apresentadas neste trabalho. Dada a natureza irregular da aplicação, podemos observar que a topologia

Figura 23 – Gráfico de taxas de *cache miss* da Aplicação LU



Fonte: Elaborado pelo Autor

Figura 24 – Média de Acessos a Memória da Aplicação LU



Fonte: Elaborado pelo Autor

foi de grande impacto na execução do algoritmo, uma vez que execuções com a mesma quantidade de clusters obtiveram resultados muito diferentes.

Dada a intensa comunicação entre os núcleos, percebe-se que aplicações com memórias *cache* menos centralizadas obtiveram piores resultados, enquanto topologia mais centralizadas tiveram resultados melhores, como a topologia Cluster.

Tabela 11 – Regras/Conceitos da Aplicação LU

Tipo	Confiança(suporte)	Dados de Acontecimento	Dados de Objetivo
Conceito	100%(20)	16 Processadores	TO / BC
Conceito	100%(20)	32 Processadores	TB
Conceito	100%(12)	AC	TB
Regra	63%(60)	-	BC
Regra	90%(21)	AMB	BC
Regra	90%(20)	32 Processadores / TB	BC
Regra	89%(19)	CMP	TP
Regra	90%(10)	16 Kb / CMP	TP

Fonte: Elaborado pelo Autor

A Tabela 11 mostra as regras e conceitos selecionados para a aplicação LU dentre as 60 simulações executadas para esta aplicação. A aplicação LU mostrou muitas regras a cerca do consumo de energia. A maioria delas para o baixo consumo especificamente. Outro detalhe foi que não foram descobertos muitos conceitos a cerca da aplicação, apesar de muitas regras com alto valor de confiança.

A Tabela 11 mostra as regras e conceitos gerados para esta aplicação. Em 100% das execuções (12 execuções ao todo) que obteve um alto consumo de potência resultam em um tempo de execução bom.

Os outros dois conceitos gerados referem-se ao consumo de potência para arquiteturas com 16 e 32 processadores. Novamente, tem-se um baixo consumo de potência para arquiteturas com 16 processadores, porém podemos ver uma relação com esses dois conceitos que quanto menor o número de processadores, melhor o tempo de execução.

A primeira regra formada afirma que em 63% das execuções temos um consumo de potência baixo. A segunda aumenta de 65% para 90% a porcentagem de execuções que obtiveram um baixo consumo quando se tem também um bom/baixo acesso a memória. Arquiteturas com 32 processadores tendem a ter um tempo de execução bom e um baixo consumo de potência.

Pensando em outras características, destacam-se também as últimas duas regras. A penúltima afirma que em 89% das execuções que apresentaram uma taxa de *cache miss* péssima, tem-se um tempo péssimo de execução. O mesmo pode se dizer para arquiteturas com 16 kB de memória, que aumenta esse valor de 89% para 90%.

Na aplicação LU, a topologia *cluster* destaca novamente com melhores resultados de desempenho pela avaliação convencional. Também é percebido que arquiteturas com mais processadores tem um tempo de processamento maior.

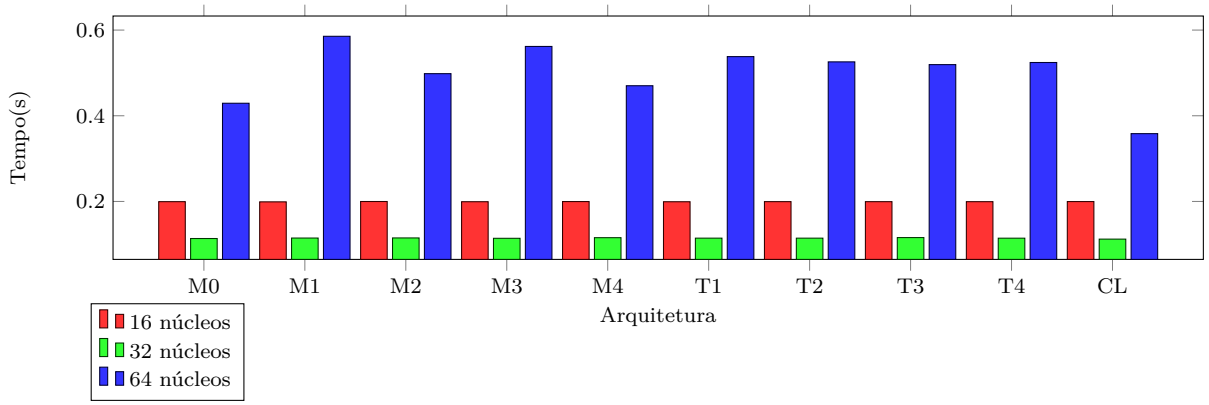
Utilizando a AFC, poucas regras e conceitos a cerca dessa aplicação foram geradas. Arquiteturas que apresentam um alto consumo possuem um tempo de execução bom e

há indícios que arquiteturas com uma taxa de *cache miss* péssima possuem um tempo de execução péssimo.

4.1.7 TSP

A aplicação TSP praticamente não muda no quesito tempo de execução médio por arquitetura. Praticamente todas as arquiteturas apresentam tempos médios de 0,2 segundos. O mesmo pode se dizer sobre a taxa de *cache miss*, onde ficaram todas próximas de 0%. Porém, o tempo médio por número de *clusters* obteve uma peculiaridade. A Figura 25 mostra os tempos de execução médio por número de processadores.

Figura 25 – Gráfico de Tempos da Aplicação TSP

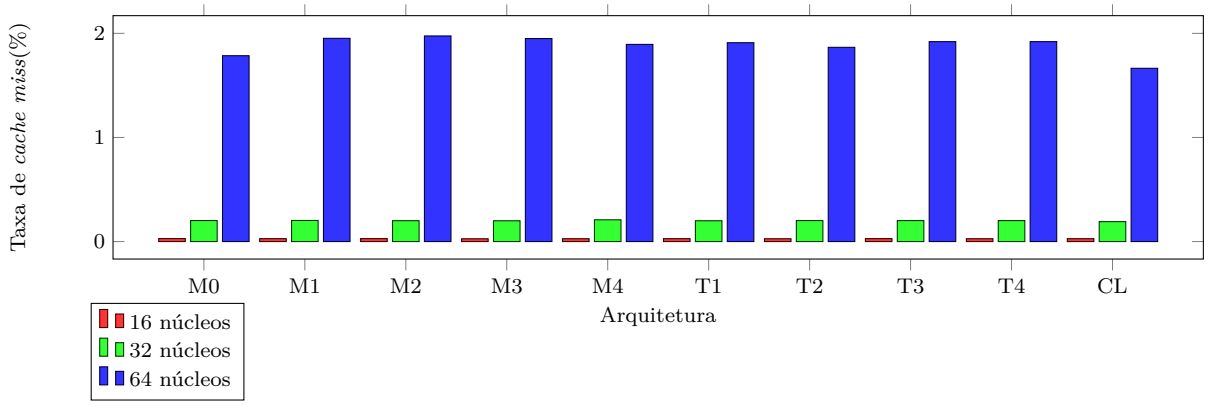


Fonte: Elaborado pelo Autor

Houve um ganho de desempenho quando o número de processadores subiu de 16 para 32. Porém, quando esse número subiu de 32 para 64, o desempenho piorou.

A Figura 26 mostra os resultados de taxas de *cache miss* da aplicação TSP.

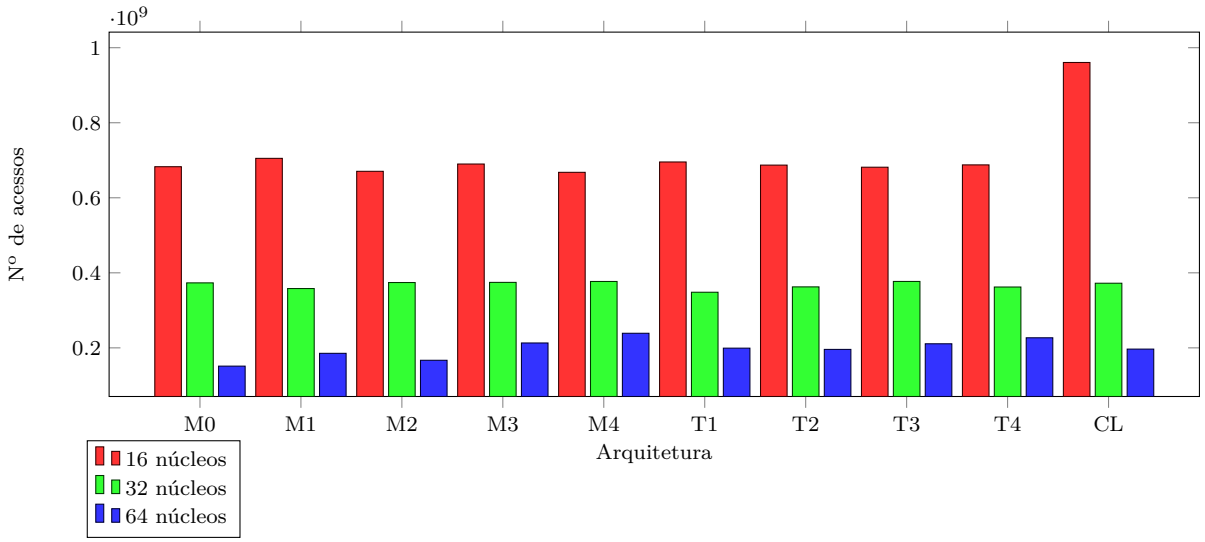
Figura 26 – Gráfico de taxas de *cache miss* da Aplicação TSP



Fonte: Elaborado pelo Autor

Pode-se perceber que o aumento no número de processadores aumentou consideravelmente a taxa de *cache miss*. Ainda que o pico seja pequeno em relação as outras aplicações (2%), os valores mínimos são próximos de 0%.

Figura 27 – Média de Acessos a Memória da Aplicação TSP por núcleo



Fonte: Elaborado pelo Autor

Se tratando de acessos a memória, como mostra a Figura 27, não temos variações pela topologia. Quanto ao número de núcleos, podemos ver que o acesso médio por núcleo é menor quanto mais núcleos a arquitetura possuir.

A Tabela 12 mostra as regras e conceitos selecionados para a aplicação TSP dentre as 60 simulações executadas para esta aplicação.

Tabela 12 – Regras/Conceitos da Aplicação TSP

Tipo	Confiança(suporte)	Dados de Acontecimento	Dados de Objetivo
Conceito	100%(40)	CMO	TO
Conceito	100%(20)	16 Processadores	TO / BC / CMO
Conceito	100%(20)	32 Processadores	TO / MC / CMO
Conceito	100%(20)	64 Processadores	AMO
Regra	67%(40)	-	TO / CMO
Regra	90%(20)	64 Processadores / AMO	CMP
Regra	60%(20)	64 Processadores / AMO	AC
Regra	78%(18)	64 Processadores / CMP / AMO	TP

Fonte: Elaborado pelo Autor

A aplicação TSP apresenta alguns conceitos e regras que conseguem definir bem o comportamento da aplicação. Muitos conhecimentos descobertos giram em torno do número de processadores da rede.

A Tabela 12 mostra os conceitos e regras gerados para essa aplicação. O primeiro

conceito afirma que 100% das execuções (40 execuções ao todo) que apresentam uma taxa de *cache miss* ótima/baixa possuem um tempo de execução ótimo.

Os próximos dois conceitos também se relacionam com o tempo de execução. O segundo conceito mostra que arquiteturas com 16 processadores possuem um tempo de execução ótimo, um baixo consumo de potência e uma taxa de *cache miss* ótima/baixa. O terceiro conceito mostra um tempo de execução ótimo, um consumo de potência médio e uma taxa de *cache miss* também ótima/baixa.

A primeira regra mostra que 67% das execuções mostram um tempo de execução ótimo e uma taxa de *cache miss* ótima/baixa. Em seguida, todas as demais regras destacadas envolvem o número de processadores na rede.

A segunda regra afirma que 90% das execuções em arquiteturas com 64 processadores que apresentam um acesso a memória ótimo obtiveram uma taxa de *cache miss* péssima/alta. A terceira regra afirma que estas mesmas regras apresentam em 60% das vezes um alto consumo de potência. E a quarta e última regra mostra que 78% das arquiteturas com 64 processadores, acesso a memória ótimo e uma taxa de *cache miss* péssima/alta apresentam um tempo de execução péssimo.

Pela avaliação convencional, a aplicação TSP também não é tão previsível. Todas as arquiteturas apresentam um tempo médio de execução idêntico por arquitetura. A peculiaridade está no fato de que o crescimento ou decrescimento do número de processadores não indicou melhora ou piora em seu desempenho.

Com a AFC, foi possível obter mais detalhes dessa aplicação. Em arquiteturas com 64 processadores, execuções com uma taxa de *cache miss* ótima mostram um tempo de execução ótimo, com um acesso à memória ótimo. Com uma taxa de *cache miss* péssima e acesso à memória ótimo, o tempo de execução é péssimo. Um ponto importante da AFC é enxergar algumas relações, como por exemplo, *cache miss* péssima e acesso à memória ótimo. Ou seja, uma solução para melhoria de desempenho na execução do TSP passa pela redução do *cache miss*, mas não pela melhoria de acesso à memória principal. Mesmo com um número que pode ser considerado alto, uma vez que pode ser menor quando da redução do *cache miss*, o acesso à memória está dentro de uma faixa considerada ótima. É claro que um pequeno crescimento nesse acesso aumenta consideravelmente o tempo médio de execução, por isso, modificações no código para melhor uso da *cache* podem ser a melhor solução para melhoria de desempenho, sem que seja necessário um investimento na melhoria da arquitetura em utilização.

4.2 Consumo de Potência

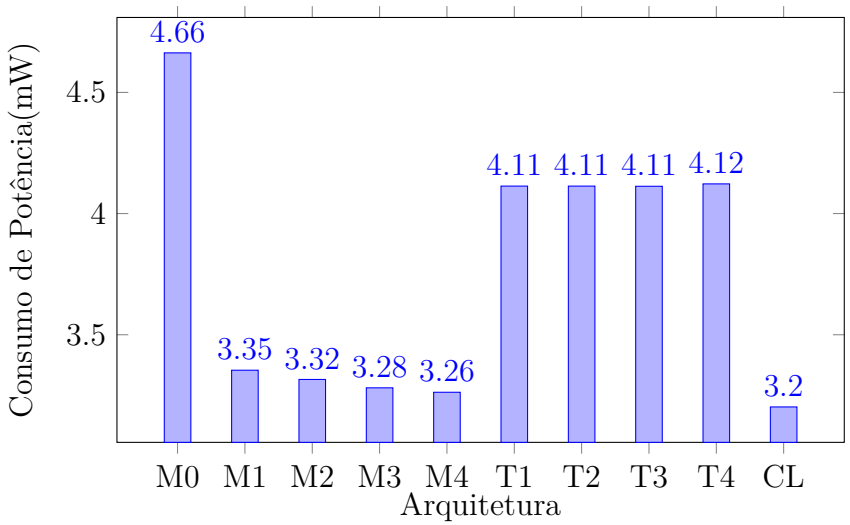
O consumo de potência de cada aplicação dependeu predominantemente pelo número de processadores presentes na arquitetura. Isso quer dizer que todas as vezes que se incluíam mais processadores nas arquiteturas e, consequentemente, outros elementos para adaptar ao novo número de processadores, o consumo de potência aumentou. O padrão, porém, seguiu-se o mesmo.

Logicamente, as arquiteturas que possuíam o menor número de elementos sempre obtiveram um consumo de energia menor. É o caso das arquiteturas em malha, excluíndo-se a M0, cujo sempre possuiu o maior número de elementos que todas as outras, uma vez que tem uma memória *cache* L2 para cada núcleo, enquanto as outras possuem uma memória *cache* L2 para cada quatro processadores.

O tamanho das memórias *cache* L2 presentes em cada arquitetura não afetou como esperado este quesito analisado, mostrando a pouca influência da capacidade da *cache* no consumo de potência do sistema, pelo menos sobre os valores utilizados nas simulações.

A Figura 28 mostra o consumo de potência da aplicação FAST, escolhida aleatoriamente, por arquitetura como exemplo de comportamento.

Figura 28 – Consumo de Potência por Arquitetura da Aplicação FAST



Fonte: Elaborado pelo Autor

Esse comportamento se segue para todas as aplicações, variando apenas os valores médios, máximos e mínimos, mas o ranking de consumo de potência se manteve o mesmo. A Tabela 13 mostra o ranking de consumo de cada aplicação.

Tabela 13 – Ranking de Consumo de Potência por Aplicação

Aplicação	Posição	Média de Consumo (mW)	Consumo Máximo (mW)	Consumo Mínimo (mW)
IS	1	3,730	9,971387	1,367079
FAST	2	3,733	9,52444	1,372838
LU	3	3,915	10,174228	1,541775
GF	4	4,153	11,756858	1,521501
KM	5	4,467	12,798341	1,626654
FN	6	5,325	14,680764	2,162585
TSP	7	7,099	12,347185	2,552016

Fonte: Elaborado pelo Autor

A aplicação IS apresentou a menor média de consumo de potência enquanto a aplicação TSP apresentou a maior média. Pode-se perceber que a posição dos picos de valores máximos e mínimos pode não condizer com a posição do consumo médio em alguns casos.

4.3 Regras e Conceitos Gerais

O objetivo do estudo apresentado nesta seção é com o intuito de classificar as aplicações, uma vez que não faz sentido comparar minuciosamente os resultados pois cada aplicação tem uma natureza diferente.

A ênfase das escolhas de conceitos e regras para entrarem nessa seção são aquelas cujo dados de acontecimento possuem pelo menos uma aplicação ou uma arquitetura, com o intuito de conseguir classificar cada um desses objetos. A Tabela 14 mostra regras selecionadas geradas por todas as 420 execuções, onde as aplicações também passam a ser parâmetros.

Tabela 14 – Regras/Conceitos da Todas as Aplicações

Tipo	Confiança(suporte)	Dados de Acontecimento	Dados de Objetivo
Conceito	100%(60)	FN	TO / CMO / AMO
Conceito	100%(60)	GF	TO / CMO
Conceito	100%(60)	KM	TO / CMO
Conceito	100%(60)	TSP	TO / CMO / AMO
Conceito	100%(20)	FAST / 32 Processadores	TB AMO
Regra	98%(60)	IS	AMP
Regra	98%(60)	IS	CHM
Regra	98%(60)	FAST	AMO
Regra	95%(60)	IS	TP
Regra	82%(60)	FAST	TB
Regra	77%(60)	LU	AMO
Regra	78%(42)	T1	AMO
Regra	76%(42)	M1	AMO
Regra	75%(42)	T3	AMO
Regra	52%(42)	M1	BC
Regra	52%(42)	M3	BC

Fonte: Elaborado pelo Autor

Pode-se perceber com a Tabela 14 em que algumas aplicações, as arquiteturas tendem a ser mais previsíveis que outras. As aplicações FN, GF, KM e TSP tem alguns padrões definidos, como tempo de execução ótimo e taxas de *cache miss* baixas em relação as demais aplicações.

Sobre as arquiteturas, pode-se perceber que há algumas regras que apontam que as arquiteturas M1 e M3 possuem uma tendência a um baixo consumo de potência. Já as arquiteturas M1, T1 e T3 aparentam ter um um acesso a memória ótimo.

Há indícios também a cerca das aplicações IS e FAST. A aplicação IS possui uma taxa de 95% de suas execuções com um tempo péssimo e 98% com uma taxa de *cache miss* e acesso a memória péssima/alta. A aplicação FAST aparece também com um acesso a memória ótimo em 98% dos casos.

4.4 AFC *versus* Avaliação Convencional

Através da avaliação convencional, o conhecimento gerado é predominantemente composto por comportamentos perceptíveis. Em sua maioria, o que ficou claro é que o número de processadores influencia bastante no desempenho das aplicações, seja ele crescente ou decrescente.

Já a AFC trouxe conhecimentos sobre seus conceitos e regras que não estavam tão claras quanto a avaliação anterior. Relações entre os dados também ficam mais claras em

algumas aplicações.

No geral, a AFC foi capaz de descobrir menos fatos que a avaliação convencional, porém mostra mais indícios. É relativamente fácil dizer qual aplicação consome mais ou menos potência, bastando apenas olhar suas médias de consumo por execução. A AFC conseguiu classificar apenas quatro das sete aplicações nesse quesito.

Porém, a AFC mostrou indícios até mesmo sobre as arquiteturas, algo que não foi possível ser feito na avaliação convencional através da forma em que foram feitas as análises. Foram descobertos indícios também a cerca de algumas aplicações. No desempenho do sistema, as regras geradas mostram quais as características influenciam no desempenho, seja o que aumenta ou diminui ou, em alguns casos em ambos. Por exemplo, na aplicação FN, a avaliação convencional não conseguiu definir um padrão de comportamento. Porém, com a AFC, regras indicaram que taxas de *cache miss* médias resultaram em um baixo consumo de energia. Em outro exemplo, a avaliação convencional mostrou que a aplicação GF possui comportamento parecido independente da arquitetura utilizada neste trabalho. Já a AFC mostra que arquiteturas com baixo acesso a memória resulta em um tempo de execução baixo, ou seja, melhor desempenho.

Além disso, avaliação por AFC evidenciou ligações complexas entre as variáveis do sistema, seja ela de configuração ou de resultado. Como exemplo, na aplicação TSP tivemos duas regras apontando que arquiteturas com 64 processadores e acesso a memória baixo resultam em um alto consumo de potência. Combinado com uma taxa de *cache miss alta*, ainda teremos um tempo de execução alto.

Na aplicação KM, as regras mostram que arquiteturas com uma taxa de *cache miss alta* combinado com um acesso a memória baixo possuem um tempo de execução bom. Porém, ao aumentar o número de recursos da arquitetura (processadores, memória *cache L2* e roteadores), o consumo de potência será alta com 80% de chances. Mas, em 56% dos casos em que o consumo de potência for alto, o tempo de execução será ótimo.

Aplicar esses conhecimentos nos algoritmos das aplicações pode melhorar o desempenho do sistema. Como os dois exemplos descritos anteriormente, na aplicação TSP, modificar o código para que se tenha uma taxa de *cache miss* baixa pode diminuir o tempo de execução.

Comportamentos também foram evidenciados pela AFC. Na aplicação TSP, por exemplo, não foi encontrado conceitos ou regras com uma confiança alta a cerca de arquiteturas com mais recursos (64 processadores) resultando em alto consumo de potência, a não ser quando combinado com outros fatores. O mesmo ocorre com a aplicação LU, onde arquiteturas com 64 processadores nem aparecem nos conceitos e regras de confiança alta. No geral, arquiteturas com 64 núcleos, obviamente, consomem mais potência. A AFC então mostrou que os picos de energia da aplicação está presente não apenas quando

há mais recursos e sim quando também há um baixo acesso a memória e uma taxa de *cache miss* alta. Mas há aplicações que apenas isso não é o suficiente para se ter picos de consumo.

A aplicação IS também trouxe regras onde mostra o resultado de uma combinação de vários dados de acontecimento. Um conceito mostra que arquiteturas organizadas em cluster, em 100% das simulações, resultaram em um tempo ótimo de execução, mesmo com uma taxa de *cache miss* alta. Porém também mostrou um conceito onde 4 fatores (32 Processadores, desempenho bom, taxa de *cache miss* alta e acesso a memória alta) resultam em um baixo consumo de potência. Este exemplo mostra que a AFC é capaz de encontrar a relação entre os dados de forma que pela avaliação convencional não seria fácil de encontrar.

Utilizados em conjunto, a avaliação convencional e a avaliação por AFC mostram resultados complementares. Regras conflitantes mostram com a avaliação convencional mostraram erros de análise. Regras que confirmam as análises servem para provar a análise feita. E as regras descobertas a cerca de novos dados revelam indícios e comportamentos não descobertas na avaliação convencional.

Variando-se apenas quatro valores (tamanho das arquiteturas em relação ao número de núcleos, tamanho das memórias *cache*, topologias de organização das arquiteturas e posição das memórias dentro da rede-em-chip) entre as simulações e analisando-se quatro resultados (tempo de execução, consumo de potência, taxa de *cache miss*), a AFC nos trouxe um conjunto de regras expressivo para entender melhor o comportamento das arquiteturas/aplicações. Porém, o número de variáveis analisadas é baixo visto a quantidade de elementos presentes dentro de cada sistema. Ainda assim, a AFC demonstrou-se capaz de encontrar padrões e destacar características que influenciam os sistemas mesmo com o baixo número de variáveis, sendo este o objetivo principal deste trabalho.

Algumas regras mostram que comportamentos binários, aqueles cujo valores contrários obtém resultados contrários, são raros ou não existem. Como um exemplo, pela Tabela 9 evidenciou uma regra que afirma que arquiteturas com 16 kB de memória *cache* (valor mínimo de memória neste trabalho), baixo consumo de potência e acesso a memória alto resultará em uma taxa de *cache miss* alta. Mas não há uma regra que evidencia arquiteturas com 64 kB de memória *cache* (valor máximo de memória neste trabalho), alto consumo de potência e acesso a memória baixo resultará em uma taxa de *cache miss* baixa.

Este fato aponta para dois comportamentos. O primeiro seria que certas características são impossíveis de se alcançar. Porém, este pensamento pode ser descartado uma vez que estão sendo utilizados valores relativos dentre as simulações. O segundo seria que mais características devem compor as regras e conceitos que giram em torno do

resultado não encontrado. Isso explica como nem todas as características estão presentes nas tabelas de resultados.

Um último fator e não menos importante que os demais apresentados até aqui está na realimentação das arquiteturas através da alteração das arquiteturas através das regras encontradas em suas execuções. Alguma regra importante evidenciada pela AFC pode levar a melhorias no sistema, porém aplicá-la e coletar novos dados para análises. Tem-se como exemplo duas regras, uma para aplicação IS e outra para a aplicação FN.

Para a aplicação IS, a uma regra que evidencia em que todos os casos que a arquitetura cluster é utilizada na simulação resultam em tempo ótimo de execução e alta taxa de *cache miss*. Outras topologias reafirmariam a escolha da topologia cluster como melhor a ser utilizada para esta aplicação ou se há alguma que mostre resultados mais interessantes.

Para aplicação FN, foi evidenciada uma regra que afirma que arquiteturas com 64 núcleos, 16 kB de memória *cache*, alto consumo de energia, uma alta taxa de *cache miss* e um baixo acesso a memória irão resultar, com 80% de chances, em um tempo de execução baixo. Sendo assim, mais simulações feitas com essas características, variando-se outras características, podem reafirmar o impacto dessa regra ou evidenciar um comportamento mais detalhado.

Por fim, as regras mostram também qual são as características que degradam o desempenho do sistema, seja através de tempo, consumo de potência ou quaisquer outras métricas. A realimentação, ou seja, aplicação de resultados em novas simulações, enriquece os dados analisados, uma vez que as regras podem ficar mais fortes e apresentar outros dados que também influenciam o sistema.

5 CONCLUSÕES

A AFC se mostrou uma técnica simples de ser utilizada, porém não completa. Houve fatos que foram identificados pela avaliação convencional sem que houvesse uma regra com confiança forte ou conceito sobre eles.

Algumas informações fáceis de deduzir, como o aumento de consumo de potência pelo aumento do número de componentes dentro de rede, foram facilmente evidenciadas pela avaliação convencional e vários conceitos e regras foram gerados a cerca deste comportamento, evidenciando sua efetividade.

A avaliação convencional também consegue mostrar alguns comportamentos evidentes devido a natureza das aplicações. Aplicações como o KM ou TSP teriam em geral um número baixo de acesso à memória pois precisam mais de processamento do que consultar a memória a todo instante. A AFC conseguiu mostrar este comportamento através de conceitos e regras.

Dados não tão claros aparecem mais frequentemente pela avaliação com AFC do que a avaliação convencional. Alguns comportamentos representados pelos conceitos e regras que não ficaram claras pela avaliação convencional.

Como exemplo, a aplicação FAST apresentou, através da AFC, que arquiteturas com memória de 64 kB por núcleo tendem a ter uma taxa de *cache miss* alta, com 90% dos resultados apresentando esta condição. Em conjunto com outra regra que implica que arquiteturas que apresentam uma taxa de *cache miss* alta, 64 processadores e acesso a memória ótimo, há um alto consumo de energia.

Outro exemplo, a aplicação GF, com 67% dos casos analisados, apontam que arquiteturas com 16 kB de memória por processador possui uma taxa de *cache miss* alta. Porém, 70% dos casos apontam que uma taxa de *cache miss* alta resultará em um bom consumo de energia.

Estas correlações entre os dados existentes não é tão evidente através da avaliação convencional, que em geral apresenta apenas relações entre duas ou três características. Já a AFC mostra várias relações envolvendo um número variável de características. E quanto mais variáveis e dados a serem analisadas pela AFC maior é a qualidade das regras

e conceitos gerados, bem como a relação entre as variáveis fica mais evidente.

Porém, a AFC fica atrelado a forma como os dados foram discretizados antes de ser aplicado. O método utilizado para a discretização fora dividir os resultados em intervalos iguais com os extremos tendo os maiores e menores valores como referência.

Assim, discretizações diferentes podem resultar em dados diferentes e, consequentemente, os conceitos e regras que foram gerados. Comportamentos diferentes poderiam ter sido obtidos apenas se trocando esses dados.

Portanto, a utilização da AFC como método de avaliação é promissora mas quando utilizada como complemento da avaliação e não como método único. Fazer uma comparação entre os resultados pela AFC e pelo método convencional confirmaria a veracidade das informações deste novo método.

5.1 Trabalhos futuros

Há mais dados que podem ser utilizados nas avaliações que não foram aproveitados neste trabalho. A utilização da AFC como método de avaliação evidenciaria outros comportamentos a cerca das aplicações e arquiteturas mostradas neste trabalho. A análise de um volume maior de dados e variáveis poderia evidenciar mais comportamentos. Por exemplo, há uma regra gerada pela AFC que mostra em que 78% dos casos em que se há 64 processadores, acesso a memória baixo e uma taxa de *cache miss* alta resulta em um tempo de execução alto. Porém, não há nenhuma regra gerada com um bom grau de confiança cujo afirma que 16 processadores, acesso a memória alto e uma taxa baixa de *cache miss* resultam em um tempo de execução baixo. Isto ocorre pois devem existir mais características que, combinadas a estes fatores, possui um tempo de execução mais baixo.

Analisar mais variáveis da simulação trará uma qualidade maior das regras e conceitos gerados pela AFC. Neste trabalho, com apenas quatro variáveis de entrada e quatro variáveis de saída, vários comportamentos ficaram evidentes. Porém, como dito anteriormente, nem todos os resultados possíveis apareceram nas tabelas. Aumentando-se o número de variáveis utilizadas irá fazer com que mais regras, relações e comportamentos apareçam nas análises.

A AFC se mostrou um método totalmente dependente da discretização dos dados antes de ser aplicada. Trocar os métodos de discretização e fazer comparações entre as regras encontradas de cada método mostrariam a influência da escolha da discretização nos resultados.

Os acessos as memórias *cache* mostraram uma influência grande na rede em alguns casos. Analisar o tráfego de dados nas redes-em-chip para correlação das topologias

e acessos às memórias podem mostrar padrões de comportamento que influenciam na ocorrência de falhas ou acertos em *cache*.

A realimentação dos sistemas simulados pode ressaltar como a AFC pode melhorar nos seus respectivos desempenhos. Uma vez que as modificações fossem feitas sobre as arquiteturas e/ou algoritmos, pode-se ter resultados melhores ou um melhor entendimento sobre o impacto de cada variável analisada.

Por fim, a AFC também mostrou diversas partes onde os códigos devem ser modificados a fim de se melhorar o desempenho de cada aplicação. Modificar as aplicações com base nas regras do AFC reduziria tanto o tempo de execução dessas aplicações ou reduzir o consumo de energia.

REFERÊNCIAS

Hela Belhadj Amor, Abbas Sheibanyrad, and Frédéric Pétrot. A distributed nuca architecture using an efficient noc multicasting support. In *Digital System Design (DSD), 2017 Euromicro Conference on*, pages 184–191. IEEE, 2017.

David H Bailey, Eric Barszcz, John T Barton, David S Browning, Robert L Carter, Leonardo Dagum, Rod A Fatoohi, Paul O Frederickson, Thomas A Lasinski, Rob S Schreiber, et al. The nas parallel benchmarks. *The International Journal of Supercomputing Applications*, 5(3):63–73, 1991.

Debajit Bhattacharya and Niraj K Jha. Analytical modeling of the smart noc. *IEEE Transactions on Multi-Scale Computing Systems*, 2017.

Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, pages 72–81. ACM, 2008.

Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

Gabriele Bleser and Didier Stricker. Advanced tracking through efficient image processing and visual–inertial sensor fusion. *Computers & Graphics*, 33(1):59–72, 2009.

Daniel Carmo, Matheus Souza, and Henrique Freitas. Avaliação de topologias de redes-em-chip usando simulação de sistemas completos e aplicações paralelas, 10 2016.

Bhavya K Daya, Li-Shiuan Peh, and Anantha P Chandrakasan. Low-power on-chip network providing guaranteed services for snoopy coherent and artificial neural network systems. In *Proceedings of the 54th Annual Design Automation Conference 2017*, page 87. ACM, 2017.

Guang Deng and LW Cahill. An adaptive gaussian filter for noise reduction and edge detection. In *Nuclear Science Symposium and Medical Imaging Conference, 1993., 1993 IEEE Conference Record.*, pages 1615–1619. IEEE, 1993.

Sérgio Mariano Dias and Newton José Vieria. Um arcabouço para desenvolvimento de algoritmos da análise formal de conceitos. *Revista de Informática Teórica e Aplicada*, 18(1):31–57, 2011.

Javier Diaz, Camelia Munoz-Caro, and Alfonso Nino. A survey of parallel programming models and tools in the multi and many-core era. *IEEE Transactions on parallel and distributed systems*, 23(8):1369–1386, 2012.

Lee Kee Goh, Bharadwaj Veeravalli, and Sivakumar Viswanathan. Design of fast and efficient energy-aware gradient-based scheduling algorithms heterogeneous embedded multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 20(1):1–12, 2009.

Paul Gratz, Changkyu Kim, Robert McDonald, Stephen W Keckler, and Doug Burger. Implementation and evaluation of on-chip network architectures. In *Computer Design, 2006. ICCD 2006. International Conference on*, pages 477–484. IEEE, 2006.

Roger W Hockney. *The science of computer benchmarking*. SIAM, 1996.

Raj Jain. *The art of computer systems performance analysis: techniques for experimental design, measurement, simulation, and modeling*. John Wiley & Sons, 1990.

Kyoung-jae Kim and Hyunchul Ahn. A recommender system using ga k-means clustering in an online shopping market. *Expert systems with applications*, 34(2):1200–1209, 2008.

Sebastian Kobbe, Lars Bauer, Daniel Lohmann, Wolfgang Schröder-Preikschat, and Jörg Henkel. Distrm: distributed resource management for on-chip many-core systems. In *Proceedings of the seventh IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis*, pages 119–128. ACM, 2011.

Shashi Kumar, Axel Jantsch, J-P Soininen, Martti Forsell, Mikael Millberg, Johny Oberg, Kari Tiensyrja, and Ahmed Hemani. A network on chip architecture and design methodology. In *VLSI, 2002. Proceedings. IEEE Computer Society Annual Symposium on*, pages 117–124. IEEE, 2002.

Hyung Gyu Lee, Naehyuck Chang, Umit Y Ogras, and Radu Marculescu. On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(3):23, 2007.

Guilherme Madalozzo, Liana Duenha, Rodolfo Azevedo, and Fernando G Moraes. Scalability evaluation in many-core systems due to the memory organization. In *Electronics, Circuits and Systems (ICECS), 2016 IEEE International Conference on*, pages 396–399. IEEE, 2016.

Vincent Nélis, Patrick Meumeu Yomsi, Luís Miguel Pinho, José Fonseca, Marko Bertogna, Eduardo Quiñones, Roberto Vargas, and Andrea Marongiu. The challenge of time-predictability in modern many-core architectures. In *14th International Workshop on Worst-Case Execution Time Analysis*, 2014.

Zhi-Liang Qian, Da-Cheng Juan, Paul Bogdan, Chi-Ying Tsui, Diana Marculescu, and Radu Marculescu. A support vector regression (svr)-based latency model for network-on-chip (noc) architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(3):471–484, 2016.

Siddheswar Ray and Rose H Turi. Determination of number of clusters in k-means clustering and application in colour image segmentation. In *Proceedings of the 4th international conference on advances in pattern recognition and digital techniques*, pages 137–143. Calcutta, India, 1999.

Howard L Rolf. Friendly numbers. *The Mathematics Teacher*, 60(2):157–160, 1967.

Amit Kumar Singh, Muhammad Shafique, Akash Kumar, and Jörg Henkel. Mapping on multi/many-core systems: survey of current and emerging trends. In *Proceedings of the 50th Annual Design Automation Conference*, page 1. ACM, 2013.

Kamna Solanki et al. Performance and energy evaluation of network-on-chip infrastructure. In *Inventive Computation Technologies (ICICT), International Conference on*, volume 3, pages 1–5. IEEE, 2016.

Matheus A Souza, Tulio T Cota, Matheus M Queiroz, and Henrique C Freitas. Energy consumption improvement of shared-cache multicore clusters based on explicit simultaneous multithreading. In *2017 International Symposium on Computer Architecture and High Performance Computing Workshops (SBAC-PADW)*, pages 1–6. IEEE, 2017.

Matheus A Souza, Pedro Henrique Penna, Matheus M Queiroz, Alyson D Pereira, Luís Fabricio Wanderley Góes, Henrique C Freitas, Márcio Castro, Philippe OA Navaux, and Jean-François Méhaut. Cap bench: a benchmark suite for performance and energy evaluation of low-power many-core processors. *Concurrency and Computation: Practice and Experience*, 29(4), 2017.

Matheus Alcântara Souza. Exploração de espaço de projeto de arquiteturas de processadores many-core baseados em redes-em-chip com uso de simulação de sistemas completos. Master's thesis, 2015.

Yi-Ran Sun, Shashi Kumar, and Axel Jantsch. Simulation and evaluation for a network on chip architecture using ns-2. In *Proceedings of the IEEE NorChip conference*, pages 167–172, 2002.

Theocharis Theocharides, G Link, Narayanan Vijaykrishnan, MJ Invin, and Vamsi Srikantam. A generic reconfigurable neural network architecture as a network on chip. In *SOC conference, 2004. Proceedings. IEEE international*, pages 191–194. IEEE, 2004.

Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. Constrained k-means clustering with background knowledge. In *ICML*, volume 1, pages 577–584, 2001.

David Wentzlaff and Anant Agarwal. Factored operating systems (fos): the case for a scalable operating system for multicores. *ACM SIGOPS Operating Systems Review*, 43(2):76–85, 2009.

Yuankun Xue, Zhiliang Qian, Guopeng Wei, Paul Bogdan, Chi-Ying Tsui, and Radu Marculescu. An efficient network-on-chip (noc) based multicore platform for hierarchical parallel genetic algorithms. In *Networks-on-Chip (NoCS), 2014 Eighth IEEE/ACM International Symposium on*, pages 17–24. IEEE, 2014.