UNIVERSIDADE FEDERAL DO PAMPA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

THIARLES SOARES MEDEIROS

OTIMIZAÇÃO DO TEMPO DE VIDA DE PROCESSADORES *MULTICORE* HOMOGÊNEOS ATRAVÉS DA VARIAÇÃO DO GRAU DE TLP E DAS POLÍTICAS DE ALOCAÇÃO DE *THREADS*

Alegrete
2021

THIARLES SOARES MEDEIROS

OTIMIZAÇÃO DO TEMPO DE VIDA DE PROCESSADORES *MULTICORE* HOMOGÊNEOS ATRAVÉS DA VARIAÇÃO DO GRAU DE TLP E DAS POLÍTICAS DE ALOCAÇÃO DE *THREADS*

Dissertação submetida ao Programa de Pós-graduação em Engenharia de Software da Universidade Federal do Pampa, como requisito parcial para obtenção do título de Mestre em Engenharia de Software.

Orientador: Arthur Francisco Lorenzon

Alegrete
2021

Ficha catalográfica elaborada automaticamente com os dados fornecidos
pelo(a) autor(a) através do Módulo de Biblioteca do
Sistema GURI (Gestão Unificada de Recursos Institucionais) .

M488o Medeiros, Thiarles Soares
Otimização do tempo de vida de processadores multicore homogêneos através da variação do grau de TLP e das políticas de alocação de threads / Thiarles Soares Medeiros.
105 p.

Dissertação(Mestrado)-- Universidade Federal do Pampa, MESTRADO EM ENGENHARIA DE SOFTWARE, 2021.
"Orientação: Arthur Francisco Lorenzon".

1. Otimização. 2. Envelhecimento. 3. Afinidade de threads. 4. Paralelismo a nível de threads. 5. OpenMP. I. Título.

THIARLES SOARES MEDEIROS

OTIMIZAÇÃO DO TEMPO DE VIDA DE PROCESSADORES MULTICORE HOMOGÊNEOS ATRAVÉS DA VARIAÇÃO DO GRAU DE TLP E DAS POLÍTICAS DE ALOCAÇÃO DE THREADS

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Software da Universidade Federal do Pampa, como requisito parcial para obtenção do Título de Mestre em Engenharia de Software.

Dissertação defendida e aprovada em: 11 de junho de 2021.

Banca examinadora:

_____

Prof. Dr. Arthur Francisco Lorenzon

Orientador

UNIPAMPA

_____

Prof. Dr. Marcelo Caggiani Luizelli

UNIPAMPA

_____

Prof. Dr. Marcelo Brandalero

UNIPAMPA

Prof. Dr. Samuel Xavier de Souza

UFRN

Assinado eletronicamente por **ARTHUR FRANCISCO LORENZON**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 14/06/2021, às 11:50, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Marcelo Brandalero**, **Usuário Externo**, em 14/06/2021, às 12:08, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **Samuel Xavier de Souza**, **Usuário Externo**, em 15/06/2021, às 06:27, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

Assinado eletronicamente por **MARCELO CAGGIANI LUIZELLI**, **PROFESSOR DO MAGISTERIO SUPERIOR**, em 15/06/2021, às 16:00, conforme horário oficial de Brasília, de acordo com as normativas legais aplicáveis.

A autenticidade deste documento pode ser conferida no site https://sei.unipampa.edu.br/sei/controlador_externo.php?acao=documento_conferir&id_orgao_acesso_externo=0, informando o código verificador **0545984** e o código CRC **E3772EDC**.

À minha esposa Marlucy e meus filhos, Lucas e Laura.

AGRADECIMENTO

Agradeço a minha família pelo apoio e compreensão pois muitos foram os momentos que não estive presente. Vocês são a minha grande motivação em tudo que faço.

Agradeço ao meu orientador e amigo Arthur. Obrigado pela confiança aceitando ser meu orientador nesse processo de evolução como pesquisador. Como amigo obrigado pelo incentivo e apoio nas minhas dificuldades. Espero ter atendido suas expectativas!

Não foram momentos fáceis, mas agradeço a todos que de alguma forma apoiaram essa etapa. Certamente, vocês contribuíram para meu progresso.

RESUMO

O avanço na tecnologia dos transistores tem permitido o aumento no número de *cores* em um único chip. Isso, por sua vez, possibilita o desenvolvimento de sistemas de alto desempenho com melhores capacidades para explorar o paralelismo no nível de *threads* (TLP). No entanto, isso também leva a problemas relacionados à temperatura desses sistemas. Alcançar altas temperaturas acelera o processo de envelhecimento dos componentes de *hardware* influenciando suas causas (por exemplo, instabilidade de temperatura de polarização negativa - NBTI). Adicionalmente, este efeito não depende apenas do número de *cores*, mas também da distância entre eles e seu uso. Além disso, as aplicações paralelas apresentam vários padrões, como irregularidade, cálculos desequilibrados ou altas taxas de comunicação. Essas diferentes características podem acentuar esses efeitos adversos. Com etapa preliminar desta dissertação, realizamos a execução de treze *benchmarks* bem conhecidos em três arquiteturas *multicore* diferentes para avaliar diferentes configurações de TLP e políticas de *placement* e estratégias de afinidade de *threads* implementadas em OpenMP. Os resultados demonstraram que não há uma configuração única que forneça o melhor NBTI para todas as aplicações. Diante deste cenário, esta dissertação propõe AATS, uma metodologia para reduzir o *aging* através da combinação de exploração de TLP e estratégia de alocação de *threads*. AATS foi empregado de duas maneiras, uma que atua de modo *offline* executando externamente à aplicação e uma *online* que faz a otimização em tempo de execução. AATS *offline* apresentou resultados efetivos na redução do *aging* com uma distância de apenas 1.02 da melhor solução encontrada na exploração de espaço de projeto para a arquitetura AMD de 16 *cores*. Ainda, é capaz de encontrar uma solução em um grande espaço de exploração avaliando um número reduzido de configurações, como por exemplo na arquitetura IBM de 160 *cores* na qual foram avaliadas, na média, apenas 1,9% das configurações possíveis. No caso do AATS *online* foi possível observar que os ajustes dinâmicos durante a execução da aplicação não possibilitaram redução do *aging*, principalmente pelo aumento do tempo de execução. A justificativa para tal está no aumento da quantidade de acessos à memória devido à variação das políticas de alocação, as quais requerem atualização dos dados que estão nas memórias *cache* quando as *threads* são realocadas.

Palavras-chave: Otimização. Envelhecimento. Afinidade de threads. Paralelismo a nível de threads. OpenMP.

ABSTRACT

The advancement in transistor technology has allowed an increase in the number of cores in a single chip. This, in turn, enables high-performance computing systems with better capabilities to exploit thread-level parallelism (TLP). However, this also leads to unforeseen issues related to the temperature of these systems. Reaching high temperatures speeds up the aging process of hardware components by influencing their causes (e.g., negative bias temperature instability – NBTI). In addition, this effect depends not only on the number of cores, but also the distance between them and their use. On top of that, parallel applications present various patterns, such as irregularity, unbalanced computations, or high rates of communications. These different characteristics may accentuate such adverse effects. With the preliminary stage of this dissertation, we perform thirteen well-known benchmarks in three different multicore architectures to evaluate different TLP configurations and thread placement policies and affinity strategies implemented in OpenMP. The results demonstrate no haves a configuration that delivers the best NBTI for all applications. Given this scenario, this dissertation proposes AATS, a methodology to reduce aging through the combination of TLP exploration and threads allocation strategy. AATS was employed in two ways, one that acts in a offline way running outside the application and a online that makes the optimization at run time. AATS offline showed effective results in reducing aging with a distance of only 1.02 from the best solution found in the design space exploration for the AMD 16 cores architecture. Still, it is able to find a solution in a large space exploration evaluating a reduced number of configurations, as example in the IBM 160 cores architecture in which, on average, only 1.9% of the possible configurations were evaluated. In the case of AATS online was possible to observe that the dynamic adjustments during the execution of the application did not make possible to reduce the aging, mainly due to the increase in the execution time. The justification for this is in the increase of the number of accesses to the memory due to the variation of the allocation policies, in which they require updating of the data that are in the cache memories when the threads are reallocated.

Keywords: Optimization. Aging. Thread affinity. Thread-level parallelism. OpenMP.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

AATS	*Aging Aware Thread Scheduling*
API	*Application Programming Interface*
BT	*Block Tri-diagonal NAS Parallel Benchmark*
CG	*Conjugate Gradient NAS Parallel Benchmark*
CPU	*Central Process Unit*
DSE	*Design Space Exploration*
DTM	*Dynamic Thermal Management*
EP	*Embarrassingly Parallel NAS Parallel Benchmark*
FT	*Discrete 3D fast Fourier Transform NAS Parallel Benchmark*
FFT	*Fast Fourier transform application*
HCI	*Hot Carrier Injection*
HMP	*Heterogeneous Multiprocessors*
HPCG	*High Performance Conjugate Gradients benchmark*
HS	Hotspot Rodinia *Benchmark*
IPP	Interface de Programação Paralela
IS	*Integer Sort NAS Parallel Benchmark*
JA	*Jacobi application*
LU	*Lower-Upper Gauss-Seidel NAS Parallel Benchmark*
LULESH	Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics application
MG	*Multi-Grid NAS Parallel Benchmark*
MOS	*Metal Oxide Semiconductor*
MPSoC	*Multiprocessor System-on-chips*
NBTI	*Negative Bias Temperature Instability*
NoC	*Network-on-Chip*
NUMA	*Non-Uniform Memory Access*
PMOS	*P-Channel Metal Oxide Semiconductor*
PO	Aplicação para solução de equações de Poisson
PU	*Processing Unit*
SC	*Stream Cluster Rodinia Benchmark*
SMT	*Simultaneous Multithreading*
SO	Sistema Operacional
SP	*Scalar Penta-diagonal NAS Parallel Benchmark*
ST	Aplicação *Stream*
TLB	*Translation Lookaside Buffer*
TLP	*Thread Level Parallelism*
UA	*Unstructured Adaptive mesh NAS Parallel Benchmark*
UMA	*Uniform Memory Access*

SUMÁRIO

1 INTRODUÇÃO

No processo de evolução da tecnologia de processamento foram criados os *chips*, os quais concentravam grandes quantidades de componentes eletrônicos, tais como os transistores que são utilizados como elementos lógicos digitais (STALLINGS, 2017). Os transistores são compostos de camadas de materiais semicondutores nos quais, quando aplicada uma tensão, passam a conduzir corrente elétrica. Com o avanço da tecnologia, os dispositivos eletrônicos estão cada vez menores dada a possibilidade de fabricar transistores com tamanho de poucos nanômetros (*nm*). Por exemplo, enquanto o Intel Pentium II usava tecnologia de 350nm e possui 7,5 milhões de transistores, o Intel Core i7 (2013) tem tecnologia de 22nm e possui 1,86 bilhão de transistores (STALLINGS, 2017). Por fim, atualmente é possível fabricar processadores com tecnologia de 5nm (LIU et al., 2020). Este avanço permitiu aumentar o número de unidades de processamento no mesmo *chip*, possibilitando dispositivos de alto desempenho (STALLINGS, 2017). Assim, os processadores puderam contar com mais unidades de processamento organizadas em *cores*. Estes processadores são conhecidos como *multicore* e podem ser dotados de bilhões de transistores.

Embora ofereça melhor desempenho e eficiência energética, esse aumento de transistores em uma mesma área do chip resulta em alta densidade de potência e consequentemente maior temperatura (CHANTEM et al., 2013). A elevação da temperatura pode causar pontos de calor (*hotspots*) e consequentemente falhas no dispositivo. Isso acontece porque alguns mecanismos de degradação são diretamente afetados pela temperatura, ou seja, quanto mais alta a temperatura, maior será a degradação gerada, reduzindo a vida útil do componente de *hardware*.

Neste sentido, um dos principais fatores de degradação é a instabilidade de temperatura de polarização negativa (*Negative Bias Temperature Instability* (NBTI)) (OBORIL; TAHOORI, 2012; LERNER; TASKIN, 2017), que consiste em um problema vital de confiabilidade em dispositivos de óxido metálico-silício (*Metal Oxide Semiconductor* (MOS)). A NBTI atua principalmente em transistores do tipo MOS de canal P (*P-Channel Metal Oxide Semiconductor* (PMOS)) e está relacionada à geração de carga de óxido positiva e às armadilhas de interface em estruturas MOS sob uma combinação de temperaturas elevadas e tensões de porta negativas (STATHIS; ZAFAR, 2006; BLAT; NICOLLIAN; POINDEXTER, 1991). Isso, por sua vez, aumenta a tensão de limiar (V_{th}), gerando efeitos adversos na corrente e no atraso de propagação, degradando o desempenho do dispositivo (SCHRODER; BABCOCK, 2003). A tensão de limiar é a tensão necessária aplicada na porta para ativação do transistor, que quanto mais degradado o transistor, mais elevada será a tensão de limiar.

O impacto da NBTI no envelhecimento do processador tornou-se mais significativo em dispositivos modernos devido à agressiva redução da dimensão do transistor (WHITE;

BERNSTEIN, 2008) e integração compacta do dispositivo. Ambos afetam fortemente a temperatura de operação, intensificando o envelhecimento do processador (GÖS, 2011). No final, este aumento na tensão de limiar pode provocar um comportamento indesejado do sistema (por exemplo, eletro-migração, ruptura dielétrica e migração de tensão) (CORBETTA; FORNACIARI, 2012) para muitas aplicações críticas, aumentando ainda mais os custos operacionais, como consumo energético e substituição de componentes. Portanto, controlar a temperatura operacional é essencial para evitar a redução da vida útil do *hardware*.

No entanto, ao executar uma aplicação paralela, a temperatura do processador tende a elevar-se conforme o número de *threads* aumenta, principalmente como resultado do aumento da atividade de chaveamento nos componentes de *hardware* (*cores* e memórias *cache*). Como a NBTI é definida em função da tensão aplicada no transistor, da temperatura, do ciclo de trabalho, do intervalo de tempo entre as medições e de parâmetros da arquitetura, na forma apresentada na Seção 2.2, o meio de tentar preservar o dispositivo é utilizar metodologias focadas em minimizar o envelhecimento (*aging)* devido à degradação sofrida através do controle do número de *threads* e políticas de alocação.

A variação do grau de paralelismo a nível de *threads* (também conhecido como *Thread Level Parallelism* (TLP), que será explicado no Capítulo 2) pode possibilitar a redução do *aging* através da redução do tempo de processamento, pois quanto menor for o tempo em que o dispositivo estiver sob condições de carga de trabalho, menor será a degradação. Porém, o alto grau de TLP não garante o menor *aging* pois quanto mais unidades de processamento estiverem ativas, maior será o aquecimento, impactando negativamente no *aging*. Considerando as arquiteturas *multicore* onde as unidades de processamento estão próximas, uma unidade mesmo sem carga de trabalho pode sofrer influência das outras que estão ativas, uma vez que o calor gerado é dissipado pelo *chip*. Logo, é necessário equilibrar o grau de TLP para que o desempenho e o calor gerado pelas unidades de processamento gerem o menor impacto possível no *aging*.

Como mencionado, unidades de processamento ativas geram calor, o qual influencia o *aging* das unidades próximas. Isto é, quando as unidades ativas estão próximas umas das outras, é gerado um ponto de calor (*hotspots*), onde a temperatura é elevada não apenas pelo processamento mas também pela proximidade delas. O resultado deste comportamento é o *aging* maior em todas as unidades de processamento. Uma forma de contornar esse problema é manter as unidades ativas o mais distante possível, para que o calor gerado não fique concentrado em uma única área. Neste sentido, estratégias de alocação de *threads* podem ser empregadas para fazer essa distribuição do processamento a fim de evitar a criação de *hotspots*. Porém, dependendo da estratégia utilizada, pode haver impacto no desempenho pois durante o processamento os dados acessados na memória são armazenados na *cache* e uma mudança de unidade de processamento pode acarretar a necessidade de nova movimentação de dados entre a memória e a *cache*.

Considerando o exposto anteriormente, analisar todas as possibilidades de alocação de *threads* e grau de TLP para executar uma dada aplicação é computacionalmente custosa, não podendo ser aplicada em tempo real (HANUMAIAH et al., 2009). Portanto, é necessário que estratégias sejam aplicadas para buscar uma configuração que forneça o menor *aging*. Isto é, desenvolver uma estratégia que permita definir o grau de TLP e políticas de alocação de *threads* que minimize o *aging*.

1.1 Objetivos

Conforme a metodologia apresentada no Capítulo 4, será explorada a variação do número de *threads* na execução de aplicações paralelas, registrando os dados necessários para analisar a influência do grau de TLP no *aging*. Para isso, é necessário entender o comportamento térmico das diferentes configurações em diferentes aplicações paralelas, já que possuem comportamentos distintos, ou seja, diferentes necessidades de comunicação e tempo de processamento. Compreender como a variação da temperatura ocorre conforme o grau de TLP possibilita elaborar uma política de gerenciamento do número de *threads* em execução simultaneamente. Assim, usando a métrica baseada no *aging* serão apresentadas duas abordagens para ajustar o número de *threads*.

Serão avaliadas as políticas de *placement* e estratégias de afinidade de *threads* do OpenMP (OPENMP..., 2018). A escolha do OpenMP é devido ao fato de ser a interface de programação de aplicações (*Application Programming Interface* (API)) mais utilizada em sistemas com memória compartilhada (KIRK; HWU, 2017). As políticas de *placement* possibilitam determinar em quais recursos as *threads* poderão ser alocadas, considerando as características do sistema, isto é, número de processadores e número de unidades de processamento. Já as estratégias de afinidade de *threads* determinam como as *threads* devem ser distribuídas nos *places* definidos pelas políticas de *placement*. Assim, é possível controlar os recursos disponíveis e a forma de distribuição das *threads* utilizando variáveis de ambiente do OpenMP permitindo ajustar conforme o objetivo desejado.

A escolha adequada da política de *placement* juntamente com a estratégia de afinidade possibilita explorar de forma mais otimizada a arquitetura. A configuração poderá ser ajustada para explorar o compartilhamento de dados via memória *cache*, otimizando o desempenho de algumas aplicações, porém gerando pontos de calor conforme a quantidade de processamento. No cenário onde a maior parte do tempo é de processamento com baixa comunicação entre as *threads* poderá ser empregado uma estratégia que distribua as *threads* pelas unidades de processamento visando não competirem por recursos (como acesso à memória, por exemplo) e a geração de calor de forma descentralizada.

Diversas abordagens na redução do *aging* já foram propostas, como poderá ser observado no Capítulo 3. A variação do grau de TLP altera a quantidade de processamento simultâneo e seu ajuste poderá ser realizado considerando a temperatura do processador. Quando o processador está com a temperatura mais elevada pode-se reduzir o grau de

TLP visando também diminuir a temperatura. A adoção de políticas de alocação de *threads* também poderá ser empregada para reduzir a temperatura, alocando as *threads* nas unidades com a temperatura mais baixa. Porém, pela revisão da literatura, a combinação de técnicas de ajuste do grau de TLP e de alocação de *threads* na redução do *aging* ainda não foi abordada.

Por fim, este trabalho propõe dois métodos para otimização do *aging* utilizado uma estratégia de ajuste do TLP combinada com políticas de alocação de *threads*. Para isso, serão definidos os recursos disponíveis para alocação (*places*) e como as *threads* serão distribuídas neles (afinidade). O ajuste será realizado a cada execução da aplicação (*offline*) ou em tempo de execução aplicado às regiões paralelas (*online*), ambos de forma transparente para o utilizador. Desta forma, aplicações já existentes também poderão utilizar a metodologia sem a necessidade de configuração, codificação e/ou compilação. Como produto final, será disponibilizada para a comunidade na forma de biblioteca compatível com OpenMP.

Diante do problema em encontrar uma configuração que possibilite reduzir o *aging* através do ajuste do TLP juntamente com a aplicação de uma política de alocação de *threads*, este trabalho, resumidamente, se propõe a:

- analisar o impacto da variação do TLP no *aging*;
- analisar o impacto das políticas de alocação disponíveis no OpenMP no *aging*;
- e, apresentar dois métodos de otimização do *aging* através do ajuste dinâmico de TLP e políticas de alocação de *threads*.

1.2 Organização do texto

No Capítulo 2 estão descritos os conceitos básicos utilizados. Primeiramente a contextualização sobre a computação paralela e seu surgimento. Após são descritas as arquiteturas paralelas conceituando os termos processador, *cores* e *threads*, apresentando as formas de paralelismo seguido da descrição das memórias *cache* e sua organização nos processadores, ainda, os tipos de acesso a memória são apresentados. Além destes, os conceitos de programação paralela são abordados, apresentando o OpenMP e os modos de alocação de *threads* implementados por ele. Por fim, é contextualizado o envelhecimento dos dispositivos eletrônicos apresentando o principal mecanismo de degradação destes.

Os trabalhos relacionados estão descritos no Capítulo 3, sendo apresentados conforme o objetivo principal. As áreas de concentração são: trabalhos que empregam técnicas para controle ou redução do envelhecimento do processador; abordagens que utilizam o controle no número de *threads* em sistemas paralelos; trabalhos que analisam o posicionamento e a afinidade de *threads* fazendo a alocação nas unidades de processamento utilizando algum tipo de estratégia; e, abordagens que utilizam o controle do número de

threads e o posicionamento e afinidade de *threads* de forma conjunta. Também é apresentada a contribuição deste trabalho diante das lacunas encontradas nas abordagens analisadas.

No Capítulo 4 está a exploração de espaço de projeto, descrevendo a metodologia utilizada no processo de avaliação das arquiteturas *multicore*, bem como o conjunto de aplicações utilizadas na avaliação. Também são descritos o processo de variação do grau de TLP e de emprego das estratégias de alocação de *threads* além da forma de monitoramento da temperatura e a forma de cálculo do *aging*. Ainda, na sequência, são apresentados os resultados experimentais iniciais, caracterizando o comportamento do *aging* das aplicações com alta e baixa taxa de comunicação e apresentando as considerações sobre cada uma delas.

A contribuição deste trabalho está descrita no Capítulo 5, na qual apresenta duas abordagens para definição das configurações que minimizem o *aging* de modo *offline* ou *online*, variando o grau de TLP e aplicação de políticas de alocação de *threads*, aplicadas às aplicações paralelas ou às regiões paralelas de uma aplicação, respectivamente.

Os resultados obtidos na utilização das abordagens *online* e *offline* estão apresentados no Capítulo 6. Primeiramente são apresentados os resultados da abordagem *offline* onde é destacada a sua eficiência. Após, os resultados da abordagem *online* são apresentados, na qual são destacadas as principais dificuldades.

No Capítulo 7 são descritas as conclusões da análise das abordagens offline e online. Neste capitulo também serão apresentadas publicações realizadas e as submissões de trabalhos relacionados à pesquisa realizada. Por fim, são apresentadas as propostas de trabalhos futuros visando aprimorar as técnicas utilizadas no trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentados os conceitos fundamentais aplicados à arquitetura de computadores, computação paralela e processo de envelhecimento de dispositivos eletrônicos. Estes conceitos caracterizam as técnicas e tecnologias empregadas ao longo do trabalho.

A Seção 2.1 deste capítulo trata da computação paralela e seus conceitos. Ela inicia pela apresentação das arquiteturas paralelas, seguido pela descrição de programação paralela (Seção 2.1.2). Os conceitos em escalonamento de *threads* estão nas Seções 2.1.3 e 2.1.4, que apresentam as estratégias de *placement* e afinidade de *threads* respectivamente. A Seção 2.2 apresenta a caracterização do envelhecimento dos dispositivos eletrônicos.

2.1 Computação Paralela

Com o avanço da tecnologia utilizada na fabricação dos processadores é possível inserir um maior número de transistores em um mesmo *chip* possibilitando processadores com múltiplos *cores* (STALLINGS, 2017). Estes processadores, chamados de *multicore*, são capazes de processar múltiplos fluxos de instruções, de acordo com a quantidade de *cores* disponíveis. Cada um desses fluxos, denominados como *threads*, possui seu próprio contexto em que há dados de controle, registradores e outras informações sobre o processamento. Assim cada *thread* pode ser executada de forma independente das demais *threads* em execução. Ainda, quando a arquitetura implementa *Multithreading* Simultâneo (*Simultaneous Multithreading* (SMT)) o número de *threads* independentes que o sistema processa é maior. Neste caso, um *core* com SMT pode processar mais de um fluxo, pois consegue lidar com os diferentes fluxos de instruções.

2.1.1 Arquiteturas Paralelas

A seguir serão descritos os componentes de uma arquitetura paralela que são fundamentais para a compreensão deste trabalho. Serão descritos os componentes relacionados a processamento e após, componentes relacionados à memória seguido dos tipos de acesso a ela.

Primeiramente, define-se que cada espaço disponível para conexão de um processador é um *slot* ou soquete (*socket*). Alguns sistemas possuem suporte a múltiplos processadores sendo denominados como sistemas multiprocessados. Os processadores são geralmente tratados como dispositivos físicos que possuem um ou mais *cores* encapsulados em um mesmo chip, também conhecido como *package*. Processadores que possuem dois ou mais cores são chamados de processadores *multicore*. Quando há apenas um *core* geralmente são apenas denominados por processador (STALLINGS, 2017). *Cores* são unidades de processamento completas, com registradores, unidade lógica e aritmética,

unidade de controle e memória *cache*.

Cada *core* pode ter a capacidade de processar um ou mais fluxos independentes de instruções, denominados como *threads*. A forma de alcançar maior grau de paralelismo através de múltiplas *threads* em *hardware* é denominada de *multithreading* e possibilita criação de múltiplos fluxos de processamento. A técnica que permite que um *core* execute ao mesmo tempo vários fluxos de instruções diferentes (*threads*) é denominada de *multithreading* simultâneo (SMT)(STALLINGS, 2017). Logo, uma *thread* em hardware pode ser equivalente a um *core* em sistemas sem SMT ou a um fluxo em um *core* com SMT. Um Sistema Operacional (SO) pode tratar cada *thread* em hardware como uma unidade de processamento lógica (ou seja, um *core* lógico). As *threads* de um mesmo *core* irão compartilhar os recursos deste (memória cache, unidade lógica e aritmética, etc.) porém cada uma terá seu conjunto de registradores.

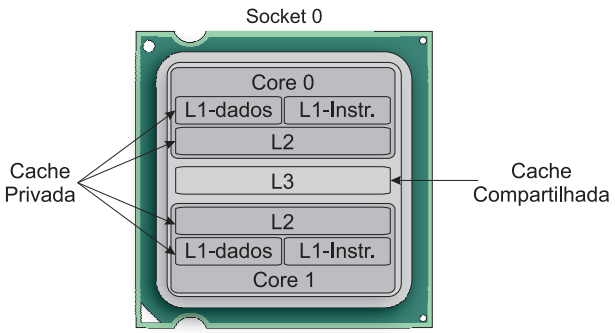
Assim, o número de *threads* de um sistema depende do número de processadores, de cores e de *threads* que cada *core* consegue processar. Ou seja, pode-se combinar multiprocessamento, *multicore* e *multithreading* em um mesmo sistema. Exemplificando, considere um sistema com dois processadores *octacore* (oito *cores*) com SMT, onde cada *core* possui duas *threads*. Neste cenário haverão 32 *threads* disponíveis pois há 16 *cores* (8+8) com duas *threads* cada. Caso não houvesse SMT o sistema teria apenas 16 *threads*, pois esse é o número de *cores* existente.

O conceito de *threads* também pode ser aplicado a nível de software onde uma *thread* é vista como a menor parte que pode ser paralelizada de uma aplicação, conforme descrito na Seção 2.1.2. Portanto, é fundamental fazer esta diferenciação ao longo do trabalho. Desta forma, para diferenciar o nível será usado o termo *threads* quando se referir a nível de *software* e sua execução. Quando se tratar do nível mais baixo, ou seja, na capacidade de processamento de fluxos independentes de instruções, será usado o termo Unidade de Processamento (*Processing Unit* (PU)). Assim, evita-se a ambiguidade que o uso do termo *threads* possa ocasionar.

Quando as PUs acessam a memória em busca de dados há um tempo de acesso que depende do tipo de conexão entre o processador e os níveis memória. Esse tempo de acesso impacta no desempenho na execução da aplicação. Para otimizar o acesso à memória os processadores possuem memórias locais para armazenar cópias dos dados, evitando a constante busca de dados na memória principal, ainda, o tempo de acesso aos dados armazenados nelas é menor (RAUBER; RÜNGER, 2010). Este tipo de memória é chamada de *cache* e pode ser implementada em diferentes níveis. Nos processadores atuais podem haver até três níveis de *cache*. O primeiro nível (L1) de *cache* é o que fica mais próximo ao processador. Este nível ainda pode-se dividir em dois tipos: *cache* de instruções e *cache* de dados. Este nível de *cache* é interna junto ao *core*. O segundo nível (L2) pode ou não estar dividida em *cache* de dados e *cache* de instruções. Além disso, também pode ou não estar implementada junto ao *core* para uso exclusivo desse

sendo então compartilhada com outro *core*. Por fim, o terceiro nível (L3) geralmente é compartilhado entre os *cores* de um processador. Na Figura 1 é apresentado um exemplo de organização dos níveis de memória cache.

Figura 1 – Exemplo da organização dos níveis de memória cache



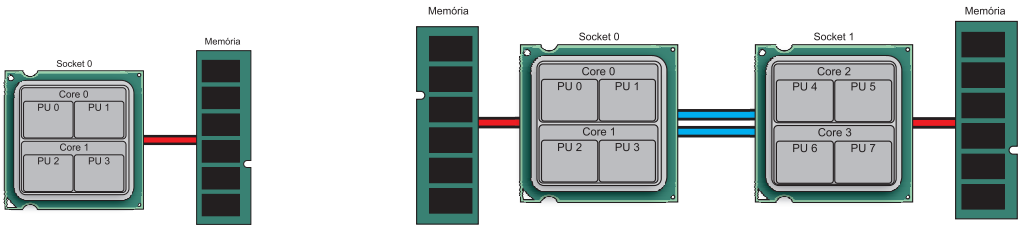
Fonte: O autor, 2021

O acesso à memória principal nas arquiteturas paralelas é caracterizado conforme o tempo de acesso à memória. Arquiteturas onde o tempo de acesso a qualquer região da memória é o mesmo são denominadas como de Acesso Uniforme à Memória (*Uniform Memory Access* (UMA)). Nas arquiteturas de Acesso Não-Uniforme à Memória (*Non-Uniform Memory Access* (NUMA)) há diferenças de tempo de acesso conforme a região da memória. Em ambos os casos, UMA e NUMA, os processadores possuem acesso de leitura e escrita em toda a memória principal.

Na arquitetura UMA, todas as unidades de processamento acessam à memória em tempo uniforme, ou seja, o tempo de acesso de um *core* é o mesmo que qualquer outro *core* do sistema. Uma arquitetura UMA é encontrada em processadores *multicore* onde todos as unidades de processamento acessam a memória principal utilizando o mesmo barramento. Quando a arquitetura é NUMA, cada processador possui uma região de memória dedicada, porém compartilhada no sistema. Cada processador terá acesso mais rápido a uma área da memória enquanto o acesso às demais áreas será mais lento. Para os processadores a memória é vista como apenas uma memória principal contínua. Quando esse faz uma requisição de uma área de memória remota (pertencente a outro processador) o barramento de comunicação entre os processadores é responsável por buscar a informação e entregar ao processador. Em sistemas multiprocessados há emprego da arquitetura NUMA. Quando necessário, os processadores se comunicam para troca de dados através de um barramento específico de alta velocidade (ex. *Intel QuickPath Interconnect - QPI* e AMD *HyperTransport*). A Figura 2 representa as arquiteturas UMA e NUMA, apresentando conceitualmente a forma de conexão dos processadores às memórias físicas do sistema.

Resumidamente, as arquiteturas paralelas são compostas por várias unidades de processamento. Estas unidades de processamento (PU) são tratadas como *threads* que

Figura 2 – Arquiteturas UMA e NUMA



(a) UMA (b) NUMA

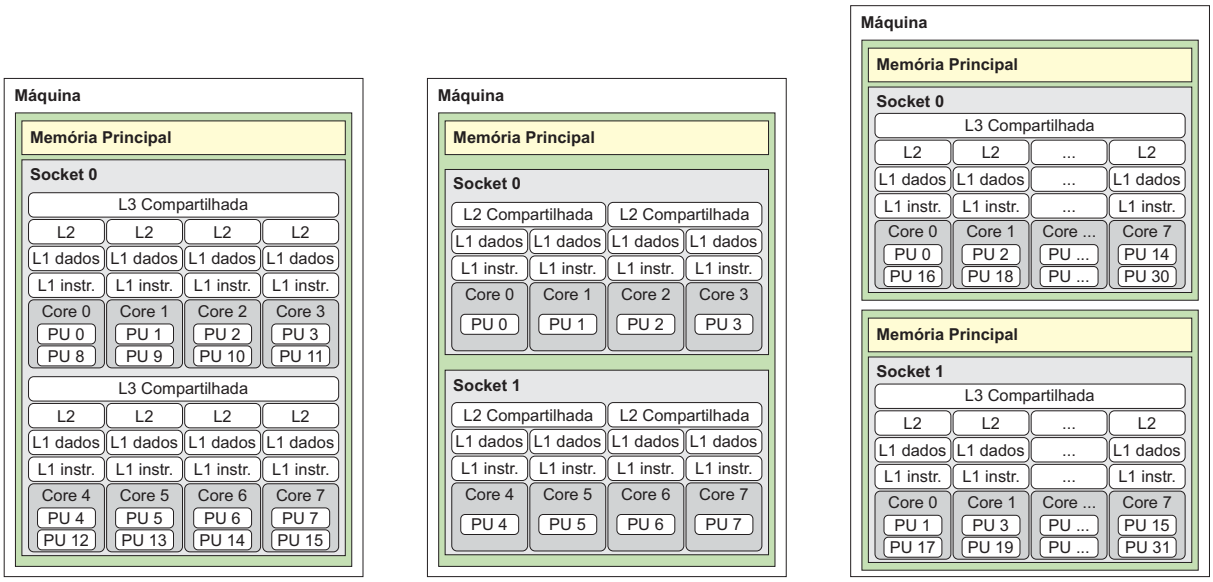
Fonte: O autor, 2021

processam fluxos de instruções. Cada *core* pode ter uma ou mais *threads* de acordo com seu projeto. Estes *cores*, podem fazer parte de um mesmo processador *multicore*, que por sua vez pode compor um sistema multiprocessado. Os *cores* tem suas memórias locais, as *caches*, para armazenar os dados utilizados que foram lidos da memória principal, visando reduzir o tempo de acesso. Ainda, dependendo da arquitetura, UMA ou NUMA, o tempo de acesso à memória principal para um processador pode variar conforme a região acessada.

Para ilustrar todos esses conceitos a Figura 3 apresenta três diferentes arquiteturas. Cada figura representa uma arquitetura com seus processadores, *cores*, PUs (*threads* de hardware) e níveis de memória *cache*. Ainda é representado a forma de acesso a memória principal, ou seja, se a arquitetura é UMA ou NUMA.

Na Figura 3a é apresentada uma arquitetura de um sistema com o AMD Ryzen

Figura 3 – Exemplos de arquiteturas paralelas.



(a) um AMD Ryzen 7 1700 (b) dois Intel Xeon E5430 (c) dois Intel Xeon E5-2640

Fonte: O autor, 2021

7 1700 com um processador de 8 *cores*. Cada core com SMT, havendo duas PUs em cada. Esta arquitetura é UMA possuindo apenas um conjunto de memória com tempo de acesso igual para todos os *cores*. Uma arquitetura com dois processadores Intel Xeon E5430 é apresentada na Figura 3b. Embora esse sistema possua mais de um processador, o acesso à memória é UMA, sendo o tempo de acesso igual para qualquer *core* do sistema. Nesta arquitetura não há SMT implementado, assim cada *core* físico é uma PU. Outro ponto que cabe destacar é que somente há dois níveis de *cache*, sendo que a *cache* L2 é compartilhada entre dois *cores*. Na arquitetura da Figura 3c é apresentado um sistema multiprocessador com dois processadores Intel Xeon E5-2640 de oito *cores* cada. Pode-se ver que o acesso à memória é NUMA, onde cada processador possui um conjunto de memória dedicada porém compõe a espaço endereçável do sistema. Ainda, tem SMT implementado, sendo que cada *core* possui duas PUs. Nesta arquitetura há 32 PUs no total.

2.1.2 Programação Paralela

Para resolver um problema computacional elabora-se um algoritmo que então é transformado em um programa ou aplicação que executa um conjunto de instruções a fim de chegar a um resultado. As instruções são executadas sequencialmente até que a aplicação termine. Porém, alguns conjuntos de instruções podem ser divididos em partes menores para serem executados concorrentemente. Estes conjuntos de instruções poderão ser executados de forma paralela, dependendo da arquitetura do sistema em que são executadas. Assim, em arquiteturas paralelas essas aplicações poderão tirar proveito do paralelismo existente.

A computação paralela explora a possibilidade de decompor um problema computacional em partes menores (*threads*[1]) que possam ser executadas simultaneamente (LORENZON; FILHO, 2019) a fim de reduzir o tempo de execução necessário para encontrar uma solução.

O conceito de *threads* conforme (STALLINGS, 2017, p. 535), *thread* é:

> [...]uma unidade de trabalho dentro de um processo que pode ser despachada. Ela inclui um contexto de processador (o qual inclui o contador de programa e o ponteiro de pilha) e sua própria área de dados para uma pilha (para possibilitar desvio de sub-rotinas). Um *thread* é executado sequencialmente e pode ser interrompido para que o processador possa se dedicar a outro *thread*.

Ainda conforme Stallings (2017), quanto mais fina a granularidade suportada pelo sistema maior a flexibilidade oferecida ao programador pois permite a paralelização em um número maior de situações.

[1] Note que neste contexto o termo *thread* está empregado a nível de *software*, conforme mencionado na Seção 2.1.1.

As aplicações paralelas são especialmente desenvolvidas para executar em ambientes com algum grau de paralelismo, onde a aplicação pode explorar o paralelismo a fim de otimizar o desempenho. Possuem características próprias, considerando a necessidade de gerenciamento das *threads* criadas como, por exemplo, a comunicação.

Para o desenvolvimento de aplicações paralelas são utilizados modelos de programação paralela que proveem uma abstração da arquitetura paralela disponível (ex. SMT, CMP, memória, etc.). Podem ser divididos em modelo de variável compartilhada para arquiteturas de memória compartilhada, e modelo de passagem de mensagem para arquiteturas com memória distribuída (DIAZ; MUÑOZ-CARO; NIÑO, 2012). Neste último, a comunicação é feita através de envio e recebimento de mensagens entre os processadores, compartilhando os dados de suas memórias locais com outros processadores (RAUBER; RÜNGER, 2010). A forma de comunicação através do compartilhamento de variáveis usa um único espaço de endereçamento acessível a todos os processadores, assim as *threads* poderão trocar dados usando variáveis compartilhadas que estão na memória principal (LORENZON; FILHO, 2019).

O OpenMP é a interface de programação de aplicações (API) mais utilizada em sistemas com memória compartilhada (KIRK; HWU, 2017). Permite ao programador criar e gerenciar aplicações paralelas utilizando diretivas de compilador, rotinas de bibliotecas e variáveis de ambiente (OPENMP..., 2018). A API OpenMP estende as linguagens C, C++ e Fortran. Pode ser dividido em duas partes: um conjunto de diretivas de compilador, que são utilizadas para definir as regiões paralelas e fazer o controle dos dados; e, outra parte que provê o controle em tempo de execução através de rotinas de biblioteca e variáveis de ambiente (OPENMP..., 2018).

As diretivas do OpenMP são definidas em C e C++ usando o mecanismo *#pragma* provido por estas linguagens. Os compiladores possuindo suporte a OpenMP irão processar as diretivas, caso contrário irão ignorá-las. Para definir uma região paralela se utiliza a diretiva *parallel*, que cria um time de *threads* OpenMP que irá executá-la.

Para controlar a execução das aplicações OpenMP é possível utilizar variáveis de ambiente para ajustar alguns parâmetros, que serão registrados em variáveis de controle interno (*Internal Control Variables* - ICV). As variáveis de ambiente disponíveis no OpenMP (OPENMP..., 2018) utilizadas ao longo deste trabalho são:

- OMP_NUM_THREADS: configura o número de *threads* a ser utilizado nas regiões paralelas;

- OMP_PROC_BIND: configura o estado inicial da ICV bind-var, utilizada para definir a estratégia de afinidade, conforme descrito posteriormente;

- OMP_PLACES: configura o conjunto inicial de *places* da ICV place-partition-var, definindo a política de *placement*, conforme descrito a seguir.

Ao executar uma aplicação OpenMP é verificado se existe a definição das PUs disponíveis, então são definidos os *places* onde as *threads* poderão ser alocadas. Os *places* são identificados por números naturais (0, 1, 2, ...) geralmente representando a menor unidade de execução do ambiente. No OpenMP o usuário poderá definir como os *places* são organizados através da variável de ambiente OMP_PLACES. A Seção 2.1.3 apresenta com mais detalhes como o posicionamento de *threads* (*placement*) é definido no OpenMP.

Quando encontrada uma diretiva *parallel* observa-se se há definição de afinidade a ser seguida. Caso não esteja definida, é utilizada a variável de ambiente *bind-var* para definir a forma de alocação das *threads*. Esta variável pode ser configurada usando a variável de ambiente OMP_PROC_BIND. A Secção 2.1.4 apresenta com mais detalhes como a afinidade de *threads* é definida no OpenMP.

2.1.3 Posicionamento de threads

As políticas de posicionamento de *threads* envolve a definição da granularidade a ser usada (*socket*, *core*, ou *threads* de *hardware*), na qual é conhecida como *placement* no OpenMP. Cada *placement* define como os *places* são definidos, sendo compostos pelas PUs dos recursos atribuídos a eles. Os *places* são utilizados na alocação de *threads* na qual são atribuídas aos *places* conforme uma estratégia de afinidade, conforme descrito na Seção 2.1.4.

No OpenMP (OPENMP..., 2018) para a especificação de *placement* de *threads* há 3 políticas pré-definidas, identificadas por nomes abstratos que as definem, sendo elas:

- ***Sockets***: O número de *places* onde as *threads* serão distribuídas é igual ao número de *sockets*;
- ***Cores***: O número de *places* é igual ao de *cores* do sistema;
- ***Threads***: Este é a granulação mais fina possível. A diferença entre esta granulação e *Cores* é que um *core* pode executar mais que uma *thread* no *hardware* quando ele implementa *SMT* (EGGERS et al., 1997).

O modo de definir os *places* em OpenMP é através da variável de ambiente OMP_PLACES que armazena uma lista de *places* separados por vírgula. Cada *place* é composto por uma lista de *threads* de *hardware* (PUs). Por exemplo, seja OMP_PLACES = '{0,1},{2,3}', neste caso há dois *places*, o primeiro com as PUs 0 e 1 e o segundo com as PUs 2 e 3.

Uma vez que a granularidade (*place*) está definida, a política de *placement* irá informar ao sistema operacional a qual *place* (*socket*, *core* ou *threads* de *hardware*) as *threads* deverão ser alocadas. As *threads* serão sempre alocadas em uma PU de um dado *place*. Como explicado anteriormente, um *core* pode ter mais de uma PU quando implementa SMT. Portanto, enquanto a política de posicionamento *threads* possui uma

relação de um para um com as PUs, as políticas *cores* e *sockets* tem uma relação de um para muitos com as PUs.

Exemplificando, considere uma arquitetura multiprocessada com dois processadores, cada um com dois *cores* com SMT. Desta forma há quatro PUs por processador totalizando oito PUs no sistema, sendo numeradas de 0 a 7. A Figura 4 representa esta arquitetura e a forma de definição dos *places*. Observe que em cada caso há a lista de *places* gerada pela política.

Na Figura 4a está representada a política de *placement Sockets*. São definidos dois *places* com quatro PUs cada. A Figura 4b apresenta a política *Cores*, definindo quatro *places* com duas PUs cada, sendo que as PUs atribuídas a um *place* fazem parte do mesmo *core*. Por fim, a Figura 4c mostra a política de *placement Threads*, totalizando oito *places* com uma PU cada.

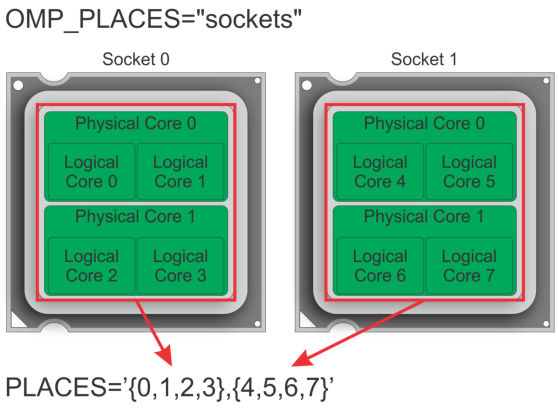
2.1.4 Afinidade de threads

A afinidade de *threads* trata do mapeamento de *threads* para *cores* como forma de otimizar o acesso à memória, considerando os níveis hierárquicos (ALESSANDRINI, 2016). Conforme apresentado anteriormente na Seção 2.1.1, pode haver tempos de acesso diferentes à memória principal, de acordo com a arquitetura do sistema, assim a localização das threads pode impactar no desempenho. Ainda, considerando os diferentes níveis de *cache*, a proximidade de *threads* que compartilham os mesmos dados pode melhorar o desempenho, já que provavelmente irão compartilhar algum nível de *cache*. O contrário também pode ocorrer, onde *threads* com independência de dados quando alocadas próximas concorrem na utilização da cache de último nível. A escolha da estratégia de afinidade impacta no desempenho e consequentemente no envelhecimento do processador, já que quanto maior o tempo de execução maior a degradação sofrida. Na Seção 2.2 o envelhecimento será discutido com mais detalhes.

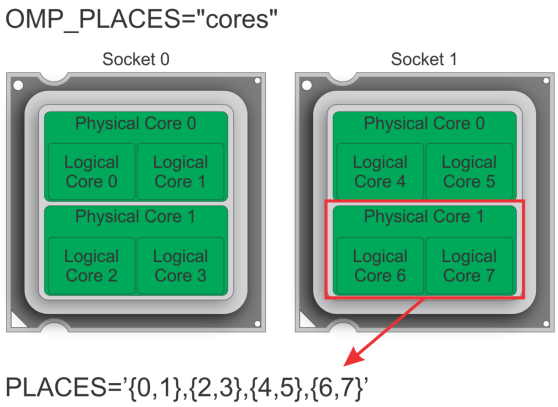
O OpenMP possui cinco estratégias distintas de afinidade de *threads* pré-definidas que são utilizadas com as políticas de *placement*, possibilitando que quando as *threads* são criadas possam ser atribuídas a *places* específicos (OPENMP. . . , 2018). São elas:

- ***Master*** As *threads* são alocadas em PUs que estão no mesmo *place* da *thread master* (ou seja, a *thread* que executa a parte sequencial, cria outras *threads* e distribui a carga de trabalho entre elas). A Figura 5 representa graficamente como a alocação de 4 *threads* ficará quando utilizadas as políticas de *placement*. A *thread master* (TM) está representada como um retângulo vermelho para caracterizar o posicionamento dela. Quando a política de *placement* é definida como *Sockets*, as *threads* são distribuídas pelas PUs que estão no mesmo *socket* da *thread master* (Figura 5a). Quando a política de *placement* é definida como *Core*, elas serão alocadas nas PUs com o mesmo *core id* (Figura 5b). E, para a política de *placement*

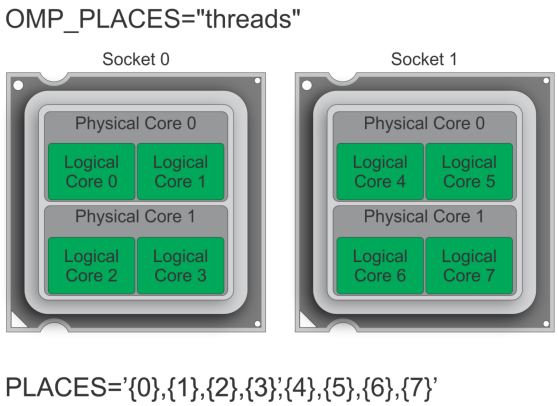
Figura 4 – Políticas de posicionamento de threads definidas no OpenMP



(a) *Sockets*



(b) *Cores*



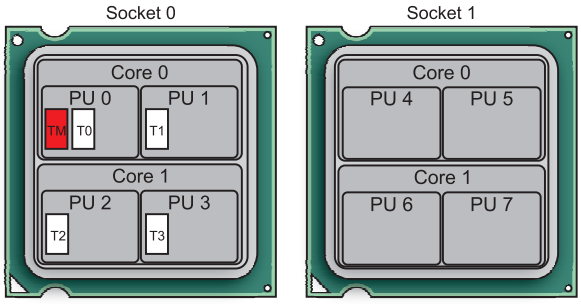
(c) *Threads*

Fonte: O autor, 2021

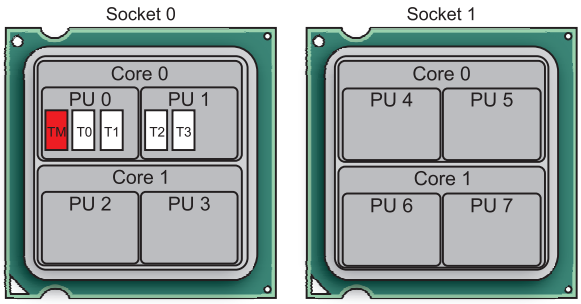
Threads, as *threads* irão compartilhar a mesma PU da *thread master* (Figura 5c).

- ***Close*** As *threads* são mantidas próximas à *thread master* em partições contíguas de *places*, ou seja, após alocar uma *thread* em um *place* a próxima *thread* será alocada no *place* subsequente. Quando o número de *threads* (NT) for maior que o de *places*

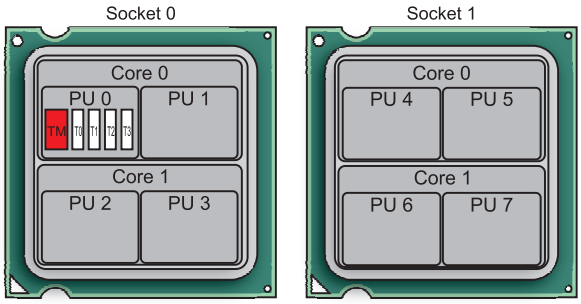
Figura 5 – Afinidade de *threads Master* combinada com as políticas de posicionamento.



(a) *Sockets*



(b) *Cores*



(c) *Threads*

Fonte: O autor, 2021

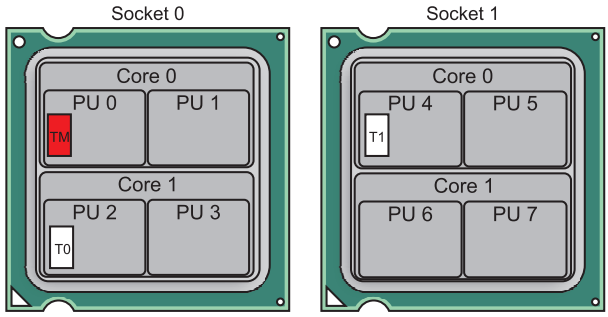
(P) grupos de *threads* serão criados com *threads* de numeração consecutiva. Cada grupo terá tamanho S_p definido por $\lfloor NT/P \rfloor \leq S_p \leq \lceil NT/P \rceil$. Então, cada grupo será alocado em um *place* consecutivo.

Quando usadas juntamente com a política *Sockets* e com o número de *threads* menor ou igual aos número de *places* ($NT \leq P$), cada *thread* em execução será alocada em uma PU de cada *socket*. Caso contrário, quando $NT > P$, mais de uma *thread* será alocada nas PUs de um mesmo *socket*.

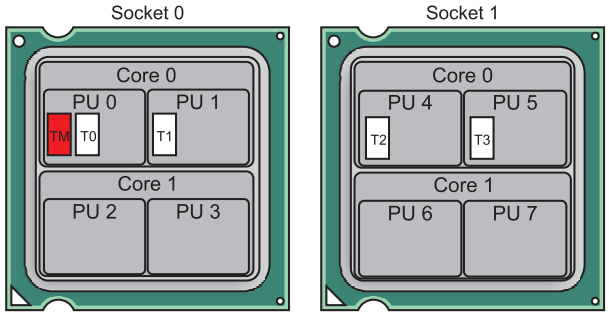
Para exemplificar considere a arquitetura apresentada na Figura 6. Para a execução de uma aplicação com duas *threads* (isto é, $NT \leq P$) cada *socket* irá receber uma *thread* que poderá ser atribuída a qualquer PU do *place*, como pode ser observado na Figura 6a. Na Figura 6b é apresentado o caso em que $NT > P$, pois são alocadas

quatro *threads* em dois *places*. Como descrito anteriormente, são criados grupos de *threads* consecutivas, ou seja um grupo com as *threads* 0 e 1 e outro com as *threads* 2 e 3. Então cada grupo é atribuído a um *socket*, pois corresponde a um *place*.

Figura 6 – Afinidade de *threads Close* combinada com a política de posicionamento *Socket*.



(a) $NT \leq P$



(b) $NT > P$

Fonte: O autor, 2021

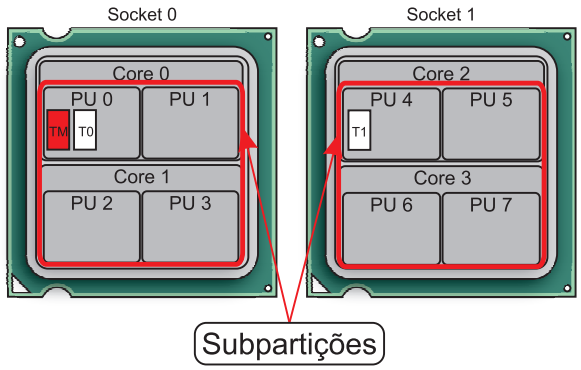
De forma análoga, quando a granularidade é definida para *Cores* e $NT \leq P$ cada *thread* em execução será alocada em uma PU de cada *core*. Quando $NT > P$, mais que uma *thread* será alocada para as PUs de um mesmo *core*.

Por fim, quando a granularidade *Threads* é definida, as *threads* em execução serão alocadas seguindo os identificadores de cada PU, ou seja, *thread* 0 na PU 0, *thread* 1 na PU 1, e assim sucessivamente.

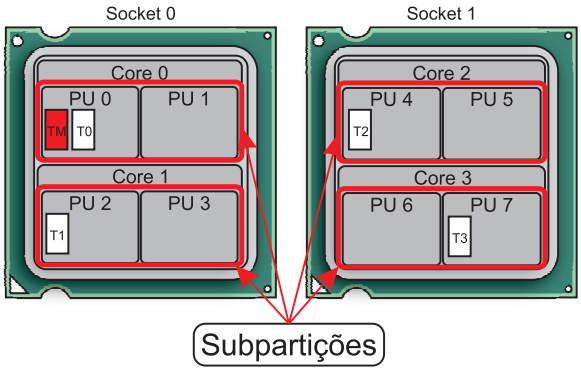
- ***Spread***: As *threads* são distribuídas de forma espalhada pelas partições de *places*. Primeiramente divide-se em subpartições, sendo o número de subpartições igual ao menor valor entre o número de threads (NT) e de *places*(P). Quando $NT \leq P$ então cada *thread* será atribuída a uma subpartição que contém $\lfloor P/NT \rfloor$ ou $\lceil P/NT \rceil$ *places* consecutivos. Ou seja, haverá NT subpartições de *places*. Assim busca-se manter o maior afastamento entre as *threads* que estão sendo alocadas.

  Caso contrário, quando $NT > P$ serão criados grupos de *threads* com números consecutivos de *threads*. Cada grupo terá tamanho S_p definido por $\lfloor NT/P \rfloor \leq S_p \leq \lceil NT/P \rceil$. Então, cada grupo será alocado em um *place* consecutivo.

Figura 7 – Afinidade *Spread* combinada com as políticas de posicionamento *Cores* e *Sockets*.



(a) *Cores* - 2 *threads*



(b) *Threads* - 4 *threads*

Fonte: O autor, 2021

Observa-se que quando $NT > P$, essa afinidade possui o mesmo comportamento da afinidade *Close*.

Como exemplo, se a granularidade for definida como *Cores* e o número de *threads* executando uma aplicação é igual a dois então serão criados duas subpartições, cada uma com dois *cores*. A primeira subpartição será composta pelos cores 0 e 1 do *socket* 0 e a segunda pelos cores 0 e 1 do *socket* 1. Então cada *thread* será alocada em uma das PUs de cada subpartição. A Figura 7a representa este cenário. Foram destacados os *places* para facilitar a compreensão. Outro exemplo é apresentado na Figura 7b, em que a afinidade *Spread* é combinada com a política *Threads*, sendo alocadas quatro *threads*. Neste caso cada uma das quatro subpartições é composta por duas PUs.

- ***False***: Afinidade de *threads* e a política de *placement* são desabilitadas e o Sistema Operacional define em qual PU uma dada *thread* irá ser alocada, além disso as *threads* poderão se mover pelas PUs disponíveis. Com essa estratégia de afinidade definida mesmo havendo a definição da forma de criação dos *places* o sistema desconsidera e aloca as *threads* da forma que julgar mais adequado.

- ***True***: O Sistema Operacional é responsável por definir em qual *place* cada *thread* ficará alocada até seu término de execução, logo uma vez que as *threads* são alocadas não serão movidas entre os *places* existentes. Cabe destacar que a *thread* pode ser movida entre as PUs que compõe o *place* em que ela está alocada.

2.2 Aging

Em busca de melhor desempenho, os processadores passaram a ter uma maior densidade lógica devido ao avanço tecnológico que possibilitou a redução do tamanho do transistor (STALLINGS, 2017). Consequentemente, há maior densidade de potência e dissipação de calor. Os efeitos térmicos sofridos nos processadores ocasionam um envelhecimento (*aging*) mais rápido. Há diversos efeitos físicos causadores do envelhecimento, tais como NBTI e o *Hot Carrier Injection* (HCI). A seguir será apresentado apenas a NBTI, pois é considerada o principal fator de degradação dos dispositivos (OBORIL; TAHOORI, 2012; LERNER; TASKIN, 2017; KIM et al., 2013).

2.2.1 Instabilidade de Temperatura de Polarização Negativa (NBTI)

Um dos principais fatores de degradação é a NBTI (OBORIL; TAHOORI, 2012; LERNER; TASKIN, 2017), que consiste em um problema vital de confiabilidade em dispositivos de óxido metálico-silício (MOS). Afeta principalmente os transistores do tipo PMOS (WHITE; BERNSTEIN, 2008) mas também pode afetar os do tipo NMOS. Com o avanço da tecnologia resultando na capacidade de fabricar transistores de tamanhos menores, ou seja, mais finos, a NBTI cada vez mais possui um impacto maior no tempo de vida do dispositivo (WHITE; BERNSTEIN, 2008). A NBTI está relacionada à geração de carga de óxido positiva e às armadilhas de interface em estruturas de MOS sob uma combinação de temperaturas elevadas e tensões de porta negativas (STATHIS; ZAFAR, 2006; BLAT; NICOLLIAN; POINDEXTER, 1991; WHITE; BERNSTEIN, 2008).

Isso, por sua vez, aumenta a tensão de limiar (V_{th}), o que terá efeitos adversos na corrente e no atraso de propagação, degradando o desempenho do dispositivo (SCHRODER; BABCOCK, 2003). A tensão de limiar é a tensão necessária aplicada na porta para ativação do transistor, que quanto mais degradado mais elevada a tensão de limiar (WHITE; BERNSTEIN, 2008; BASOGLU; ORSHANSKY; EREZ, 2010), ou seja, mais alta é a tensão de limiar tornando os dispositivos mais lentos (CORBETTA; FORNACIARI, 2012). Este atraso é inversamente proporcional a diferença entre as tensões de operação e de limiar pois para ativar um dispositivo haverá um tempo maior para elevar a tensão. Desde modo, com o passar do tempo (logo maior degradação) a diferença entre as tensões diminui o atraso aumenta.

O impacto da NBTI no envelhecimento do processador tornou-se mais significativo em dispositivos modernos devido à agressiva redução da dimensão do transistor (WHITE;

BERNSTEIN, 2008) e integração de dispositivo compacta. Ambos afetam fortemente a temperatura de operação, intensificando o envelhecimento do processador (GÖS, 2011). No final, este aumento na tensão de limiar pode provocar um comportamento indesejado do sistema (por exemplo, eletromigração, ruptura dielétrica e migração de tensão (CORBETTA; FORNACIARI, 2012)) para muitas aplicações críticas, aumentando ainda mais os custos operacionais. Portanto, controlar a temperatura operacional é essencial para evitar a redução da vida útil do hardware.

Como o efeito da NBTI mais claro de se observar é a variação da tensão de limiar necessária para a ativação do transistor. Os modelos são unânimes na utilização da variação da tensão de limiar para quantificar a degradação gerada pela NBTI (WHITE; BERNSTEIN, 2008).

Alguns modelos do comportamento do NBTI já foram propostos, tais como por White e Bernstein (2008) e Henkel et al. (2013). Neste trabalho será utilizada o modelo apresentado nos trabalhos de Lee, Shafique e Faruque (2018) e Oboril e Tahoori (2012). Este modelo compreende dois momentos diferentes da influência do NBTI: *stress* e recuperação. O primeiro, é caracterizado pelo efeito da aplicação da tensão negativa sobre o dispositivo, degradando diretamente. Já no segundo momento, após um período de *stress*, há uma recuperação parcial do material dos dispositivos reduzindo o efeito do NBTI. Assim, a modelagem utilizada deve compreender ambos momentos, para uma avaliação mais precisa da degradação ao longo do tempo, considerando que o dispositivo poderá operar por alguns anos até que o dispositivo se torne inoperante.

Conforme Lee, Shafique e Faruque (2018) a equação da variação da tensão de limiar (ΔV_{th}) em um instante $t > 0$ é dada pela Equação 2.1.

$$\Delta V_{th} \leq \int_0^1 A_N \cdot u(V_{dd}) \cdot \frac{(v(T_C) \cdot \delta_C \cdot \delta_e \cdot t_m)^n}{w(\delta_C \cdot \delta_e, T_C, t)^{2n}} d\delta_e; \tag{2.1}$$

Onde, A_N é uma constante dependente da tecnologia, T_C é a temperatura do *core* em Kelvin (°K), V_{dd} é a tensão de operação, t_m é o período de tempo analisado e δ_e é o ciclo de trabalho efetivo.

As funções $u(V_{dd})$, $v(T_C)$ e w utilizadas na Equação 2.1 são definidas a seguir.

$$u(V_{dd}) = (V_{dd} - V_{th}) \cdot exp\left(\frac{V_{dd} - V_{th}}{E_0}\right) \tag{2.2}$$

A Equação 2.2 representa a função dependente das tensões de operação (V_{dd}) e de limiar (V_{th}).

$$v(T_C) = \xi_4 \cdot exp\left(\frac{-E_a}{kT_C}\right) \tag{2.3}$$

O impacto da temperatura está descrito na função $v(T_C)$ descrita na Equação 2.3.

$$w = 1 - \left(1 - \frac{\xi_1 + \sqrt{\xi_3 \cdot v(T_C) \cdot (1 - \delta(t)) \cdot t_m}}{\xi_2 + \sqrt{v(T_C) \cdot t}}\right)^{\frac{1}{2n}} \tag{2.4}$$

Na Equação 2.4 w caracteriza a dependência do período de recuperação do dispositivo, ou seja, aquele em que não se encontra sobre *stress*.

Sendo que n, E_0, ξ_1, ξ_2, ξ_3 e ξ_4 são constantes dependentes da tecnologia, $E_a = 0,49eV$ é a energia de ativação e $k = 8,617x10^{-5}eV/K$ é a constante de Boltzman.

Analisando o impacto das variáveis ciclo de trabalho (δ_e), temperatura (T) e tensão de operação (V_{dd}) observa-se que o maior impacto no *aging* ocorre quando a tensão de operação está elevada sendo agravada quando a temperatura ou o ciclo de trabalho aumentam. Isso pode ser visto na Figura 8 na qual estão apresentados os gráficos de ΔV_{th} devido a variação do V_{dd} para as temperaturas de 25°e 90°C (Figura 8a), variação da temperatura com a tensão de operação em 0,85V e 1,4V (Figura 8b) e variação do ciclo de trabalho na condição de temperatura igual a 25°C e tensão de operação de 0,85V e 1,4V (Figura 8c). Para as Figuras 8a e 8b o ciclo de trabalho considerado é de 70%.

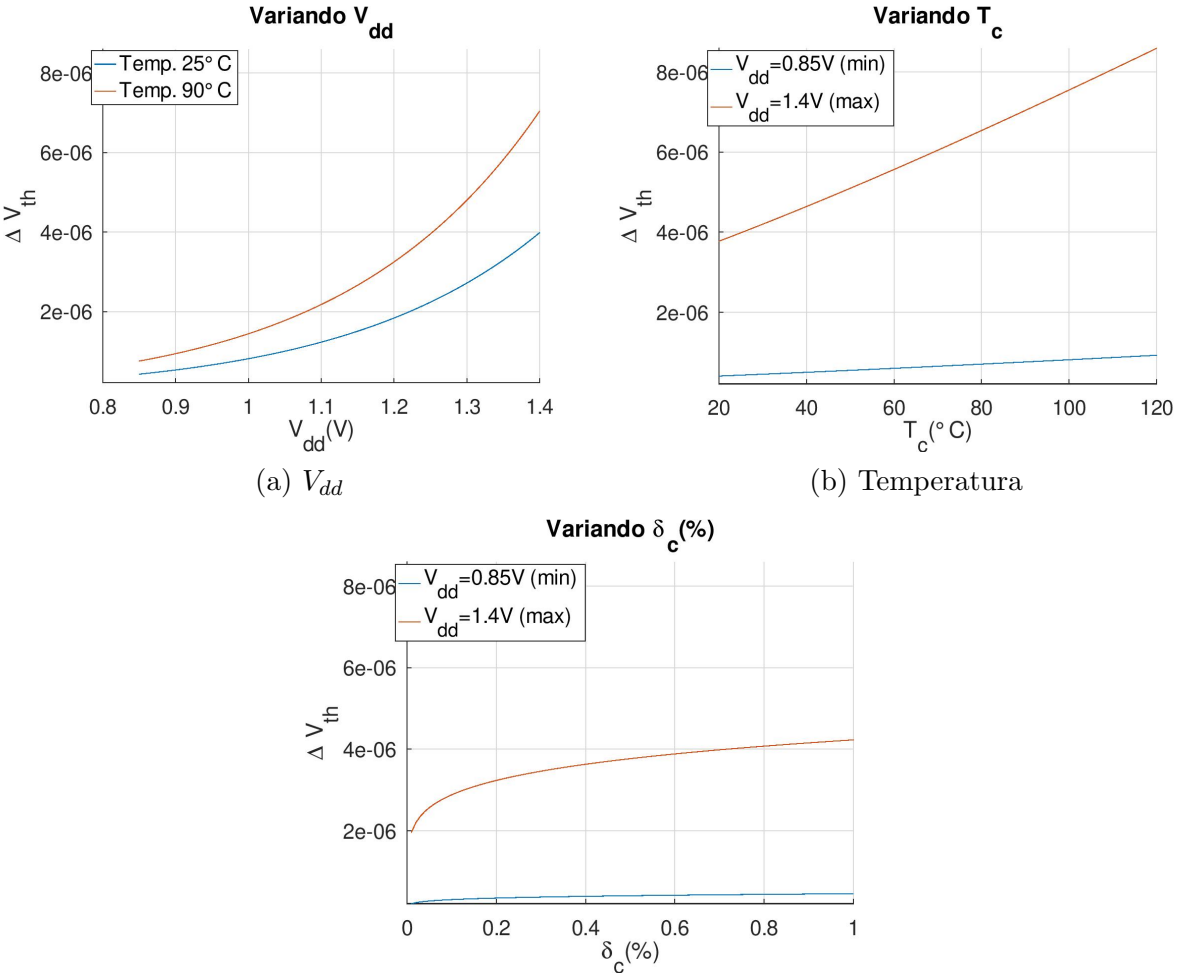
2.3 Considerações do Capítulo

Neste capítulo foram discutidos os conceitos fundamentais necessários à compreensão da pesquisa desenvolvida. Foram apresentadas de forma geral a computação paralela e suas características e o processo de envelhecimento sofrido por dispositivos eletrônicos.

A computação paralela está presente nos processadores atuais possibilitando ganho de desempenho. É caracterizada pela execução simultânea de instruções em uma arquitetura com suporte a tal. Essas arquiteturas podem suportar o paralelismo de diferentes formas, seja pela existência de múltiplos processadores, múltiplos *cores* num mesmo processador ou ainda via SMT, onde um *core* pode suportar mais de uma *thread* em hardware. Além disso, as arquiteturas podem apresentar mais de uma dessas formas de paralelismo ao mesmo tempo. Complementarmente, este capítulo abordou a organização das memórias *cache* nos processadores *multicore* e também classificou as arquiteturas quanto ao tempo de acesso à memória, isto é, UMA e NUMA.

Conceitos de programação paralela foram abordados, apresentando o conceito de *thread* a nível de *software*. Para o programador poder usufruir das capacidades das arquiteturas paralelas, modelos de programação devem ser empregados. O modelo de programação de variável compartilhada na qual é implementado no OpenMP será o contexto desse trabalho. Quanto ao gerenciamento das *threads* foram apresentados os conceitos de *placement* e afinidade. No primeiro, são definidos os locais (*places*) onde poderão ser alocadas as *threads*, já a afinidade define como as *threads* serão distribuídas nestes *places*.

Figura 8 – Comparativo do impacto no *aging*.



(a) V_{dd} (b) Temperatura

(c) Ciclo de trabalho

Fonte: O autor, 2021

Do ponto de vista da gestão do *hardware*, especificamente da gestão do tempo de vida de um dispositivo eletrônico, foi apresentado o mecanismo que possui grande influência na degradação, e consequente redução da vida útil, o NBTI. O impacto mais notável é a variação da tensão necessária para ativação de um transistor PMOS, que com a degradação do dispositivo é necessário uma tensão maior de ativação. Inevitavelmente, em algum momento o *hardware* não será capaz de fornecer a tensão necessária, ocasionando em comportamentos não desejados do sistema.

Neste cenário, vários trabalhos foram propostos para atuar no paralelismo a nível de *threads*, na afinidade de *threads* e no *aging*. No capítulo seguinte, os trabalhos relacionados à pesquisa serão apresentados e comentados. Diversas técnicas e abordagens foram propostas com diversos objetivos. Porém, como será mostrado não utilizam da combinação de TLP e políticas de *placement* e afinidade com objetivo de otimização do *aging*.

3 TRABALHOS RELACIONADOS

Conforme o exposto anteriormente, pode-se explorar o paralelismo a nível de *threads* em arquiteturas paralelas, porém questões de placement e afinidade de *threads* devem ser consideras pois podem impactar no desempenho devido ao tempo de acesso à memória ou no *aging* pois podem gerar pontos de calor no sistema. Neste capítulo são apresentados os trabalhos desenvolvidos pela comunidade acadêmica que empregam algum tipo de técnica relacionada à estas questões.

Na Seção 3.1 são apresentadas abordagens cujo principal objetivo é avaliação e gerenciamento do envelhecimento do processador, ou seja, *aging*. Trabalhos que utilizam a variação do número de *threads* e mostram a diversidade da forma de aplicação são apresentados na Seção 3.2. Na Seção 3.3 apresenta-se estratégias de *placement* e afinidade de *threads* que exploram as características das arquiteturas *multicore* e do relacionamento entre as *threads*. Combinações entre técnicas que utilizam TLP e políticas de *placement* e afinidade de *threads* são descritas na Seção 3.4. Por fim, na Seção 3.5, descreve-se as contribuições deste trabalho.

3.1 Aging

Abordagens cujo principal objetivo é avaliação e gerenciamento do *aging* do processador, que empregam diferentes técnicas visando controlar ou evitar a degradação dos sistemas serão apresentadas a seguir.

Mulas et al. (2009) propõe uma política para balanceamento térmico em arquiteturas *Multiprocessor System-on-chips* (MPSoC) através do processo de migração de *threads* com impacto mínimo na qualidade de serviço (Quality of Service – QoS). Para a migração de *threads* são considerados os custos de energia e desempenho. O objetivo é manter a temperatura dos processadores próxima a temperatura média definindo limiares inferior e superior. Desta forma, quando a temperatura de um processador atinge o limiar superior, há migração de *threads* para o processador com menor temperatura, e no outro caso quando o limiar inferior for atingido, a política migra *threads* do *core* mais quente para o que está mais frio. Assim, os parâmetros de limiar são constantemente atualizados e a temperatura média também. A política foi capaz de fazer o balanceamento térmico do sistema sem impactar na qualidade de serviço, mantendo as temperaturas dos recursos em até 3°C de diferença a temperatura média.

Corbetta e Fornaciari (2012) analisam o impacto de diferentes políticas de alocação no envelhecimento do processador e propõe uma nova. Além disso, o trabalho propõe uma metodologia para estimar a degradação gerada pelos efeitos do NBTI. A metodologia é dividida em três estágios: no primeiro, é caracterizada a arquitetura do processador; no segundo, as instruções são emuladas a fim de obter estatísticas das unidades funcionais; e por fim, a degradação é estimada. Os experimentos realizados mostram que a política

de alocação por fazer o balanceamento a nível das unidades funcionais pode reduzir a degradação em até 20% comparada às políticas *Round-Robin* e *Random-Selection*.

Zoni e Fornaciari (2013) apresentam uma abordagem para mitigar a degradação gerada pelo NBTI em *buffers* de *Network-on-Chip* (NoC). Para isso, o trabalho considera a variabilidade do processo de fabricação e parâmetros microarquiteturais. Duas técnicas são apresentadas: a primeira, que considera que não há sensores de NBTI, utiliza uma política *round-robin* para selecionar qual *buffer* recuperar. Já a segunda técnica considera que existem sensores de NBTI que possibilitam identificar quais *buffers* estão mais degradados. A estratégia de recuperação dos *buffers* é através da redução do período de estresse via *power gating*. Os resultados apresentados mostram que é possível mitigar o NBTI, principalmente quando utilizada a técnica que faz uso de sensores de NBTI.

Namazi et al. (2018) apresentam uma proposta de mapeamento de tarefas que busca manter o nível de confiabilidade com desempenho e temperatura média como fatores secundários. A proposta foi avaliada em NoCs, considerando a vida útil de cada *core*. A proposta utiliza grafos para determinar o mapeamento de tarefas e como será o comportamento quando ocorrer uma falha, resultando em migração das tarefas atribuídas ao *core* com falha. Ainda, em tempo de execução pode ocorrer a migração de tarefas de acordo com as predições definidas na fase inicial, prevenindo a ocorrência de falhas. A análise dos resultados foi dividida em duas partes, a primeira com relação à etapa de mapeamento das tarefas e a segunda etapa de predição de falhas e migração de tarefas. A primeira etapa apresentou um aumento da confiabilidade principalmente com tamanhos de malha maiores. A segunda etapa possibilita o menor impacto no tempo de vida em até 4.76%.

Reghenzani et al. (2018) apresentam um gerenciador térmico como um controlador orientado a dados que pode ser aplicado a qualquer processador. O controlador é baseado na técnica de controle de busca extrema (*Extremum-Seeking* (ES), uma técnica que não requer conhecimento prévio do problema otimizado), onde a temperatura da *Central Process Unit* (CPU) é utilizada como fator de restrição. Além disso o controlador utiliza outros dados da CPU para fazer o controle térmico. A simulação realizada, para um conjunto de parâmetros definidos, mostra que é possível realizar o controle térmico variando a máxima utilização da CPU, podendo aumentar o tempo de vida do processador.

3.2 Exploração do Paralelismo a Nível de threads (TLP)

Alguns trabalhos utilizam a variação do número de threads para otimizar diferentes tipos de métricas. As características das aplicações variam de diferentes formas, tornando complexa a busca por uma solução ideal. Abaixo, são apresentados trabalhos desenvolvidos nesta temática, onde pode-se evidenciar a grande aplicabilidade da exploração do TLP nas arquiteturas *multicore*.

Suleman, Qureshi e Patt (2008) propõe o *framework* FDT (Feedback-Driven Th-

reading) que define o número de *threads* a partir de informações do sistema. Para isso, o FDT analisa o comportamento da aplicação e de acordo com o comportamento (ou seja, usando mais largura de banda ou sincronismo de dados) então define o número de *threads* adequado que otimiza o desempenho e consumo de energia. Ainda, apresenta duas abordagens de otimização, uma para aplicações que requerem mais largura de banda e outra para aplicações caracterizadas pelo sincronismo de dados. Ambas possuem fase de treinamento onde a aplicação é executada somente por algumas iterações. Os resultados mostram que as abordagens propostas são capazes de definir o melhor número de *threads*, e quando combinadas mantém suas características fornecendo em média uma redução de 17% no tempo de execução e 59% na energia consumida, quando comparada com a modo convencional onde cria *threads* conforme os *cores* disponíveis.

Lee et al. (2010) apresentam uma abordagem em que as *threads* criadas poderão ser combinadas a fim de reduzir a comunicação ou remover sincronismos desnecessários. Para isso, primeiramente a técnica verifica os recursos disponíveis do sistema e cria partições para balancear a carga de trabalho. Estas partições são criadas a partir de dados previamente obtidos (metadados) da execução da aplicação, com foco em reduzir a necessidade de comunicação e sincronismo entre as *threads*, podendo haver união de *threads* quando possível, ou seja, *threads* combinadas. Quando a aplicação é executada, as chamadas de criação de *threads* são interceptadas pelo compilador. Então, o compilador cria as novas *threads* e aloca nas *threads* combinadas anteriormente. No geral os resultados mostram que a técnica proposta apresenta potencial de utilização, com benefício médio de desempenho de 19% em dispositivos reais em relação à quando o número de *threads* é igual ao de *cores*.

Li et al. (2010) apresentam uma metodologia para economia de energia para aplicações híbridas implementadas em MPI/OpenMP. Esta metodologia divide o processamento em fases. Cada fase é caracterizada por meio de predições de tempo, concorrência e consumo energético. Com estas informações a metodologia define qual configuração possibilita reduzir o consumo energético de cada fase, ajustando o número de *cores* e *threads* por *core* para cada fase. De forma complementar, é utilizado a variação da tensão e da frequência para explorar ainda mais o potencial de economia de energia nas fases. Para os casos testados apresentou economia de energia em relação à quando utilizando todos os processadores e todos os *cores* de um nodo, porém em escalas diferentes conforme as características das aplicações.

Sridharan, Gupta e Sohi (2014) apresenta *Varuna*, um sistema que ajusta o grau de paralelismo para melhorar utilização dos recursos disponíveis otimizando métricas de eficiência. *Varuna* monitora as condições de operação e define o grau de paralelismo de acordo com a dinâmica do programa. Para fazer o controle, utilizou-se do conceito de ta-

refas virtuais, que são mantidas em um *pool*[1] para serem executadas de forma coordenada, conforme o grau de paralelismo definido e os recursos disponíveis. Os resultados dos experimentos realizados mostraram que em média, comparados ao modo provido no *kernel* do Linux, foi possível reduzir o tempo e o consumo energético nos ambientes utilizados, reduzindo em até 33% e 32% respectivamente.

Alessi et al. (2015) propõe uma extensão do OpenMP que permite diferentes tipos de otimização, por exemplo tempo e energia, possibilitando que o programador defina o tipo de otimização por região da aplicação. A abordagem implementa um construtor (*construct*) OpenMP que permite definir as regiões de código, e duas cláusulas (*clauses*) que são utilizadas para definir a otimização desejada e ajustar parâmetros da aplicação (dependentes do código do usuário). Os testes mostraram que foi possível reduzir o consumo de energia do regulador *ondemand* (regulador de frequência padrão do Linux) mantendo a qualidade de serviço, com resultados mais expressivos em plataformas móveis.

Khdr, Amrouch e Henkel (2018) apresenta uma técnica para otimizar o desempenho determinando o número de *threads* considerando restrições no envelhecimento. A técnica define o espaço de exploração de acordo com o ΔV_{th}^{m} especificado como limite de envelhecimento. Desta forma busca a configuração que maximize o desempenho atendendo a carga de trabalho. Tabelas de relacionamento entre temperatura limite e envelhecimento são criadas como referência. Comparado com o trabalho de Mercati et al. (2017) obtiveram média de 47% de ganho de desempenho, com o máximo de 73% .

Vieira et al. (2019) analisou o comportamento de vários algoritmos implementados em três diferentes interfaces de programação paralela (Interface de Programação Paralela (IPP)), avaliando o impacto no envelhecimento de processadores embarcados. Para isso, foram avaliadas as IPPs PThreads, OpenMP, e MPI utilizando nove *benchmarks* de diferentes domínios além de avaliar três níveis de paralelismo (2,4 e 8 *threads*). Concluíram que para aplicações *memory-bound*, o OpenMP apresenta o menor envelhecimento do processador porém o mesmo não ocorre em aplicações *CPU-Bound*, levando a conclusão que a escolha da IPP correta para cada tipo de aplicação permite a redução do envelhecimento do processador significativamente (até 68%).

Geras, proposto por Medeiros et al. (2019), apresenta uma estratégia de otimização do número de *threads* que reduz o envelhecimento do processador. A abordagem utilizada não requer alterações nas aplicações OpenMP para que otimizar o envelhecimento. Geras utiliza uma estratégia de busca baseada no algoritmo *Hill Climbing* para buscar o número ideal de *threads* para uma região paralela. Uma vez definido o número de *threads*, o algoritmo monitora a carga de trabalho e constatando variação acima do esperado (no trabalho foi considerado 20%) é refeita a análise do número ideal de *threads*, fazendo o ajuste em tempo de execução. Os resultados apresentam que Geras, reduz o

[1] Um *pool* é uma estrutura lógica que registra todos os itens juntos, podendo ser compartilhado por diversos serviços.

envelhecimento na maioria dos casos testados, sendo que no melhor caso apresentou 99% de otimização comparado ao Varuna (SRIDHARAN; GUPTA; SOHI, 2014).

3.3 Estratégias de Placement e Afinidade de Threads

Estratégias de *placement* e afinidade podem explorar as diferentes características das arquiteturas *multicore* no processo de otimização. Capacidade de processamento, memória *cache* e temperatura são fatores que podem ser utilizados para definir onde e como alocar as *threads* nos recursos disponíveis (*threads* de *hardware* ou lógicas). A seguir, trabalhos que consideram a localidade e/ou afinidade das *threads* são descritos.

Sawalha, Tull e Barnes (2011) propõe dois mecanismos de escalonamento que consideram as fases de execução para mapear as *threads* para *cores* heterogêneos. Para otimizar o mapeamento, a abordagem utiliza informações de desempenho das fases que já ocorreram durante a execução. Basicamente o que difere os mecanismos é a forma de registro destas informações. Um deles, quando uma fase sem avaliação de desempenho ocorre, registra-se em uma tabela a assinatura do escalonador (dados do escalonamento de uma fase da aplicação), no outro, o registro é feito da mesma forma porém avalia o desempenho para cada tipo de *core*. Os resultados mostram que o primeiro mecanismo apresenta melhores resultados que o segundo mecanismo e que a abordagem proposta por Kumar et al. (2003), porém necessitando de mais dados para reutilizar no escalonamento.

Cruz et al. (2011) apresenta um método para arquiteturas NUMA que mapeia e aloca *threads* das aplicações com o objetivo de reduzir os custos de comunicação entre *threads* que compartilham dados. Para isso, é feita uma pré-análise da aplicação para identificar os dados compartilhados entre as *threads* gerando uma matriz de compartilhamento. Então, uma heurística é aplicada para encontrar o mapeamento das *threads* usando um grafo para relacioná-las, identificando onde a comunicação é maximizada. Quando comparado ao sistema operacional, e dependendo da característica da aplicação (compartilhamento de dados e tipo de acesso à memória), obteve até 75% de melhora do desempenho, porém em alguns casos não apresentou benefício principalmente em aplicações com comunicação homogênea ou alta carga de processamento e baixo compartilhamento de dados.

Chantem, Hu e Dick (2011) apresenta uma formulação para atribuir e escalonar tarefas em MPSoCs visando reduzir o pico de temperatura do chip. Utiliza programação inteira mista (*Mixed-Interger Linear Programming* – MILP) na formulação do problema. Para isso, a formulação considera a organização física do dispositivo na modelagem do problema. Utiliza busca binária para encontrar a configuração de atribuição e o escalonamento de tarefas que minimize o pico de temperatura. De forma complementar pode inserir um *delay* entre a execução de duas tarefas consecutivas quando a temperatura estiver próxima ao máximo aceito, com o intuito de baixar a temperatura do *core*. Demonstrou que a proposta reduz o pico de temperatura em média 10,09°C e até 30,75°C,

quando comparado com o método de economia de energia (MILP com foco em energia). Ainda, comparado ao método de redução do pico de potência (MILP com foco em potência) reduz o pico de temperatura em média 8,98°C e até 23,25°C

Em outro trabalho, Chantem et al. (2013) apresenta outra abordagem porém com foco na confiabilidade. A proposta busca aumentar o tempo de vida do sistema realizando a atribuição e escalonamento de tarefas em tempo de execução. O algoritmo define a atribuição das tarefas maiores primeiro, buscando os *cores* com menor consumo de energia buscando balancear os efeitos de degradação. A estratégia de escalonamento classifica cada *core* conforme sua degradação, e então definir quais tarefas serão atribuídas. A técnica se mostrou efetiva principalmente em sistemas com poucos *cores*, no caso demonstrado eram 4, quando comparado a algoritmos representativos descritos pelos autores e a abordagem de Huang et al. (2011). Quando o número de *cores* aumentou para 9 os resultados foram menos expressivos, sendo em alguns casos até pior.

Khdr et al. (2014) apresentam uma técnica de *Dynamic Thermal Management* (DTM) que utiliza de dois preditores e duas centrais de decisão para migrar as tarefas. Ambos preditores são empregados em cada um dos *cores*. Um dos preditores faz a análise da temperatura de *threshold* evitando que seja alcançada, Para isso utiliza o histórico e tendências de temperatura do *core*. O outro analisa o comportamento e faz o balanceamento térmico entre os *cores*, também utiliza dados das médias de temperatura e tendências térmicas do *core*. As centrais de decisão são responsáveis por utilizar as informações dos preditores e agir proativamente. Elas fazem a migração de tarefas entre os *cores* conforme necessário, prevenindo atingir a temperatura de *threshold* e minimizando a variação térmica no processador. A técnica foi capaz de reduzir até 22% a variação térmica comparado com a técnica proposta por Yeo, Liu e Kim (2008).

Diener et al. (2016) apresenta uma extensão do kMAF (técnica apresentada por Diener et al. (2014)), um *framework* que automaticamente mapeia *threads* e dados de aplicações paralelas. Nesta extensão uma nova política de mapeamento de *threads* foi proposta, na qual armazena menos dados de utilização da memória pelas *threads* e adota o algoritmo *EagerMap*, usado para mapeamentos em tempo real e com maior velocidade que outros algoritmos. O kMAF atua em tempo de execução, monitorando o comportamento do compartilhamento da memória e usando o *EagerMap* define em qual unidade de processamento cada *threads* deve executar. Com isso, é capaz de reduzir o consumo energético e melhorar o desempenho comparado ao sistema operacional, ao mecanismo de balanceamento NUMA do *kernel* do Linux, a um mecanismo simples de mapeamento de *threads* (USING..., 2012) e ao mecanismo Carrefour (DASHTI et al., 2013).

Chien e Chang (2016) propõe uma técnica de escalonamento baseada na temperatura dos *cores* e nas características das *threads* a fim de minimizar pontos de calor e manter a temperatura estável. As *threads* são classificadas conforme seu comportamento térmico, então de acordo com a classificação são atribuídos a *cores* específicos, por

exemplo, se classificadas como *threads* quentes (*threads* que apresentam potencial para aumenta a temperatura do *core*) devem ser atribuídas a *cores* que estão com a temperatura mais baixa, o contrário também ocorre nos casos de *threads* frias. O método proposto permite que em tempo de execução sejam ajustadas as alocações das *threads* caso o comportamento térmico varie. Ainda, de forma reativa pode desativar *cores* migrando as *threads* para os *cores* que permaneceram ativos. A técnica, comparada ao modo original de alocação, pode reduzir a temperatura de 2°C a 5°C.

Tu et al. (2017) apresenta uma abordagem que faz o escalonamento de tarefas baseada no envelhecimento dos *cores* em NoCs. É proposto um *framework* que utiliza a degradação causada pelo NBTI na estimativa de envelhecimento. Como forma de representação das tarefas é utilizado um grado e o objetivo é reduzir o *makespan*. Como heurística utiliza otimização de enxame de partículas, para percorrer o grafo e encontrar como as tarefas devem ser alocadas considerando o envelhecimento além de minimizar o tempo necessário. A proposta apresentou uma média de 12,3% de redução do *makespan*.

Mück et al. (2017) propõe um *framework* chamado ADAMANt, que faz o mapeamento de tarefas considerando energia, desempenho e envelhecimento em arquiteturas com processadores heterogêneos (*Heterogeneous Multiprocessors* (HMP)). A primeira etapa do algoritmo proposto é fazer a leitura dos sensores e contadores de *hardware*. Com base nas informações coletadas é feita a predição do comportamento de cada tarefa, estimando o desempenho e consumo de energia. Então, busca fazer primeiramente o mapeamento das tarefas que apresentam maiores restrições para os *cores* que possam atender os requisitos de desempenho e envelhecimento. Estes passos são repetidos ao longo da execução das tarefas, pois a técnica proposta busca equalizar o envelhecimento dos *cores*. Nos resultados apresentados foi demonstrado que em uma arquitetura *big.LITTLE* houve um aumento de 2 vezes no tempo de vida do processador em relação ao escalonador GTS do Linux modificado para considerar o envelhecimento.

Rozo et al. (2018) propõe um *framework* tolerante a falhas. Utiliza uma camada intermediária, o Nível de Confiabilidade (*Reliability Level* – RL), as tarefas são alocadas em RLs e depois é feito o mapeamento dos RLs para *cores* em tempo de execução. Desta forma possui uma parte estática durante o mapeamento das tarefas para os RLs e, uma parte dinâmica durante o mapeamento dos RLs para *cores*. Para o escalonamento utiliza um algoritmo genético que considera o grafo de tarefas, conjunto de *cores* e a estimativa da taxa de falhas. Considera as estatísticas das falhas para ordenar os *cores* que irão receber tarefas. A técnica se mostrou capaz de manter baixo a contagem de falhas conforme aumento da taxa de falhas do sistema principalmente.

Rathore et al. (2019) propõe uma estratégia que utiliza aprendizado por reforço para diminuir o envelhecimento. A técnica é chamada de *LifeGuard*. Faz o mapeamento de tarefas para *cores*, baseado nos requisitos de desempenho. Caracteriza uma aplicação através de sua execução em um conjunto de *cores*, registrando os dados de potência e

temperatura. A partir destes dados são criados grupos de potência e temperatura na qual as aplicações são classificadas. Como primeiro passo para escalonamento de tarefas são criados grupos de *cores* e então as tarefas são atribuídas aos grupos de *cores*. Uma vez que todos os *cores* de um grupo possuem tarefas atribuídas novas atribuições serão realizadas no próximo grupo. Os grupos são ordenados baseado na frequência dos *cores*. No segundo passo é aplicado o aprendizado por reforço onde fará a identificação em qual dos *cores* as tarefas devem ser mapeadas. Os resultados mostram que a técnica é capaz de aprimorar a frequência de até 74% dos *cores* comparado a HiMap (RATHORE et al., 2018). Ainda mostraram que considerando desempenho do tempo de vida a proposta é equivalente às outras abordagens comparadas, apresentando resultados que aumentam o tempo de vida em mais de 7 meses comparado a HiMap.

3.4 Exploração do TLP, Estratégias de Placement e Afinidade de threads

Algumas abordagens utilizam a combinação da exploração do TLP com estratégias de *placement* e políticas de afinidade de *threads*. Os trabalhos mostram que estas técnicas quando combinadas podem apresentar bons resultados.

Shafik et al. (2015) apresenta uma abordagem para redução do consumo de energia para aplicações OpenMP com base em um *power budget* específico e no controle térmico. Dado o *power budget* ele cria quantas *threads* forem possíveis e então, utilizando as métricas obtidas para as *threads* e *cores*, é feita a alocação. Durante a execução é monitorado o consumo de energia e caso necessário ajusta o ambiente, variando o número de *threads* e alocação. A abordagem foi comparada ao regulador de frequência do Linux *ondemand*, à técnica descrita por Cochran et al. (2011) e ao trabalho de Curtis-Maury et al. (2008). Os resultados mostraram que a técnica apresenta redução de energia de até 15% comparada às outras abordagens relatadas no trabalho.

Sensi, Torquati e Danelutto (2016) propõe NORNIR, que automaticamente configura a execução de aplicações sem considerar dados prévios de outras execuções. Tem como finalidade atender a requisitos de consumo energético e de desempenho. Uma configuração definida pelo algoritmo é composta por número de *threads*, posicionamento das *threads* e DVFS. A variação de configuração ocorrerá em tempo de execução conforme a mudança da carga de trabalho. O algoritmo divide-se em duas fases, na primeira a aplicação é monitorada e o modelo é calibrado até que a predição de desempenho e consumo energético tenha um erro aceitável. Na segunda fase, utilizando o modelo de predição já calibrado é possível predizer desempenho e consumo energético da aplicação nas configurações possíveis. O algoritmo apresenta um tempo de calibragem maior que as outras abordagens (Li e Martinez (2006) e Mishra et al. (2015)), o que pode estrar refletido na comparação da perda de desempenho que ficou em até 5% e com perda no consumo energético em até 10% quando comparados à solução ótima.

Danelutto et al. (2017) faz uma análise do comportamento de aplicações quanto ao

desempenho e consumo energético, avaliando a variação do número de *threads* e estratégias de afinidade. Os resultados apontam que políticas de afinidade em arquiteturas com SMT apresentam menor perda em cenários de desempenho e consumo energético na maioria dos casos testados. Do ponto de vista da eficiência, considerando que o ideal é o menor o número de recursos utilizados, evidenciou que as técnicas que não utilizam SMT requerem menos recursos. Por último avaliou o esforço de programação onde relata que as técnicas de afinidade são mais facilmente utilizadas quando comparadas às técnicas que alteram o número de *threads*, justificando o fato que neste último uma variação pode requerer adequação dos contextos das *threads* mantendo correta a execução da aplicação.

3.5 Contexto da Dissertação

A Tabela 1 apresenta o resumo dos dados sobre os trabalhos relacionados mencionados anteriormente. Cada coluna representa a classificação dos trabalhos quanto às abordagens realizadas, ou seja, *aging*, TLP e Políticas de *Placement* e Afinidade de *threads* (PeA).

Diante do contexto apresentado, nota-se que parte dos trabalhos está focado na exploração do TLP. Obviamente, esta área tem recebido atenção devido ao crescente número de *cores* disponíveis nos processadores. Assim, é necessário o gerenciamento da quantidade de *threads* em execução, seja por questões de desempenho, energia ou *aging*. Nesse último caso, poucos trabalhos apresentam a otimização do número de *threads* com objetivo de reduzir o *aging*.

Os trabalhos mencionados que exploram a utilização de políticas de *placement* e estratégias de afinidade se destacam por haver maior interação com o controle do *aging*. Porém, a maioria não apresenta estratégias de otimização do TLP e quando houve, não apresentou relação com o controle do *aging*.

Como pode-se observar, nenhum dos trabalhos relacionados avalia a exploração do TLP combinado com políticas de *placement* e afinidade de *threads* no gerenciamento do *aging*. Portanto, este trabalho explora a lacuna existente, avaliando o impacto de diferentes níveis de paralelismo, afinidade de *threads* e políticas de *placement* juntos no gerenciamento do *aging*. A partir desta análise entende-se que é necessário uma técnica que otimize o *aging* do processador através da definição do TLP, do *placement* e da afinidade de *threads* adequados às arquiteturas e aplicações executadas.

Tabela 1 – Resumo dos trabalhos relacionados

Referência	*Aging*	TLP	PeA
(MULAS et al., 2009)	X	-	-
(CORBETTA; FORNACIARI, 2012)	X	-	-
(ZONI; FORNACIARI, 2013)	X	-	-
(REGHENZANI et al., 2018)	X	-	-
(KHDR; AMROUCH; HENKEL, 2018)	X	X	-
(VIEIRA et al., 2019)	X	X	-
(MEDEIROS et al., 2019)	X	X	-
(CHANTEM; HU; DICK, 2011)	X	-	X
(CHANTEM et al., 2013)	X	-	X
(KHDR et al., 2014)	X	-	X
(CHIEN; CHANG, 2016)	X	-	X
(TU et al., 2017)	X	-	X
(MüCK et al., 2017)	X	-	X
(NAMAZI et al., 2018)	X	-	X
(SULEMAN; QURESHI; PATT, 2008)	-	X	-
(LEE et al., 2010)	-	X	-
(LI et al., 2010)	-	X	-
(SRIDHARAN; GUPTA; SOHI, 2014)	-	X	-
(ALESSI et al., 2015)	-	X	-
(SHAFIK et al., 2015)	-	X	X
(SENSI; TORQUATI; DANELUTTO, 2016)	-	X	X
(DANELUTTO et al., 2017)	-	X	X
(SAWALHA; TULL; BARNES, 2011)	-	-	X
(CRUZ et al., 2011)	-	-	X
(DIENER et al., 2016)	-	-	X
(ROZO et al., 2018)	-	-	X
(RATHORE et al., 2019)	-	-	X
Proposta	**X**	**X**	**X**

Fonte: O autor, 2021

4 EXPLORAÇÃO DE ESPAÇO DE PROJETO

Neste Capítulo, apresentamos a exploração de espaço de projeto (*Design Space Exploration* (DSE)) realizada com a execução de treze aplicações em três arquiteturas *multicore* com características distintas. Para isso, consideramos diferentes estratégias de *placement*, afinidade de *threads* e grau de exploração de paralelismo. Neste sentido, a Seção 4.1 descreve a metodologia utilizada nos experimentos enquanto que a Seção 4.2 discute os resultados preliminares obtidos.

4.1 Metodologia

A metodologia utilizada na caracterização do problema e na avaliação da proposta a ser apresentada será descrita a seguir. Na Seção 4.1.1 serão apresentadas as arquiteturas utilizadas. Para avaliar o comportamento do *aging* foram utilizados os *benchmarks* descritos na Seção 4.1.2 para padronizar a carga de trabalho. Cada *benchmark* foi executado com um conjunto de parâmetros de TLP e políticas de *placement* e afinidade, definidos nas Seções 4.1.3 e 4.1.4, respectivamente. Durante a execução foram coletados dados do comportamento térmico das arquiteturas conforme descrito na Seção 4.1.5, para então calcular o *aging* seguindo a metodologia apresentada na Seção 4.1.6.

4.1.1 Arquiteturas Multicore

Três distintas arquiteturas foram utilizadas na análise e validação da proposta, conforme descrito na Tabela 2. Este conjunto de arquiteturas compreende diferentes características, fundamentais para a análise da influência no processo de *aging* e posteriormente na validação da proposta apresentada. Dentre as arquiteturas utilizadas, há com diferente organização da memória principal sendo dos tipos UMA e NUMA. Há também com e sem SMT implementado, e quando implementado possuem diferentes quantidades de *threads* em cada *core* (por exemplo, duas ou quatro *threads* no caso da arquitetura IBM). Ainda, as arquiteturas são de diferentes fabricantes, sendo AMD (*Advanced Micro Devices*), Intel e IBM (*International Business Machines*).

A variedade de arquiteturas proporciona analisar diferentes situações possíveis, avaliando o comportamento em sistemas com múltiplos processadores ou com apenas um processador, o impacto da quantidade de PUs de cada *core* (SMT) e da variação da quantidade de *cache* L3.

4.1.2 Conjunto de Benchmarks

Foram utilizadas treze aplicações paralelas implementadas em C/C++ e com uso da interface de programação paralela OpenMP para exploração do paralelismo no nível de

Tabela 2 – Arquiteturas *multicore* utilizadas

Característica	AMD Ryzen 7 1700	Intel Xeon E5-2640	IBM Power9 8335-GTH
Microarq.	Zen	SandyBridge	POWER9
Nº de Sockets	1	2	2
Nº Cores Fís.	8	16 (8+8)	40 (20+20)
Nº Threads de HW (PUs)	16	32	160
Cache L3 (total)	16 MB	40 MB	400 MB
Memória Principal	16 GB	64 GB	512 GB
Acesso à Memória	UMA	NUMA	NUMA
Freq. Base CPU	3,0 GHz	2,0 GHz	3,0 GHz
V_{dd}	0,8 - 1,4V	0,6 - 1,30V	0,71 - 1,1V
Temp. Max. da CPU	95°C	75°C	80°C
Sistema Operacional	Linux - Kernel v.4.15		

Fonte: O autor, 2021

threads. Cada aplicação foi compilada com GCC/G++ 9.1, usando a *flag* de otimização -O3. São as aplicações:

- **Oito *kernels*** do conjunto do *NAS Parallel Benchmark* (BAILEY et al., 1991; JIN; FRUMKIN; YAN, 1999) implementadas em C por Seo, Jo e Lee (2011), compilados para problemas classe C:

  - *Block Tri-diagonal* (**BT**): implementa um algoritmo para resolver equações de Navier-Stokes tridimensionais através de um sistema com blocos tridiagonais de 5x5 que são resolvidos sequencialmente em cada dimensão.

  - *Conjugate Gradient* (**CG**): utiliza um método de gradiente conjugado para calcular uma aproximação de sistemas particulares de equações lineares.

  - *Embarrassingly Parallel* (**EP**): estima os limites de desempenho para operações de ponto flutuante.

  - *Discrete 3D fast Fourier Transform* (**FT**): calcula usando a transformada rápida de Fourier (FFT) um sistema tridimencional de equações diferenciais parciais, aplicando a FFT em cada dimensão separadamente.

  - *Lower-Upper Gauss-Seidel* (**LU**): resolve equações de Navier-Stokes em tres dimensões pela divisão em blocos de sistemas triangulares superior e inferior.

  - *Multi-Grid em uma sequência de malhas* (**MG**): calcula a solução de sistemas de equações de três dimensões com malhas, variando entre malhas mais finas e mais grossas.

  - *Scalar Penta-diagonal* (**SP**): implementa um algoritmo para resolver sistemas de equações através da aproximação de Beam-Warming, separando as dimensões e resolvendo sequencialmente em cada uma com um sistema penta-diagonal.

  - *Unstructured Adaptive mesh* (**UA**): provê um método de avaliação de performance como se estivesse executando aplicações que não possuem um padrão de acesso a memória (FENG et al., 2004).

- **Quatro aplicações de diferentes domínios:**
  - *Fast Fourier transform* (**FFT**) conforme descrito por Petersen e Arbenz (2004): a transformada rápida de Fourier converte do domínio do tempo para o domínio de frequência (CORMEN et al., 2009).
  - Iteração do método *Jacobi* (**JA**) conforme descrito por Quinn (2004): é um algoritmo para resolver sistemas de equações lineares de pequenas dimensões ou para sistemas grandes com grande porcentagem de entradas de zero.
  - STREAM (**ST**) conforme descrito por McCalpin (1995): é um *benchmark* simples para avaliar a largura de banda da memória e taxa de computação, trabalhando com conjuntos de dados maiores que a capacidade das memórias *cache*.
  - **HPCG** (*High Performance Conjugate Gradients*) conforme descrito por Dongarra, Heroux e Luszczek (2015): ferramenta para avaliar sistemas computacionais através de soluções de equações diferenciais parciais (DONGARRA; HEROUX; LUSZCZEK, 2015).

- **Uma aplicação real: LULESH**2.0, usado por uma variedade de simulações computacionais de problemas científicos e de engenharia que requerem modelagem hidrodinâmica (HYDRODYNAMICS. . . , 2011).

As aplicações foram classificadas utilizando duas métricas:

i) Grau de TLP: a forma utilizada para avaliar o grau de TLP de uma aplicação é a proposta por Flautner et al. (2000). Considera o tempo em que as PUs estão executando alguma parte da aplicação, calculando a média de *threads* concorrentes ao londo de todo o tempo de execução. O TLP calculado utilizando essa abordagem "é uma indicação de eficiência no uso dos recursos em um chip multiprocessador"(BLAKE et al., 2010, p. 304, tradução nossa).

ii) Taxa de acerto da *cache* L3: menor taxa de acerto da cache indica que as *threads* podem estar utilizando conjuntos de dados diferentes o que pode indicar uma maior demanda de comunicação entre as *threads*. Essa métrica pode ser utilizada para estimar a necessidade de comunicação entre *threads* que não compartilham a mesma *cache* L3.

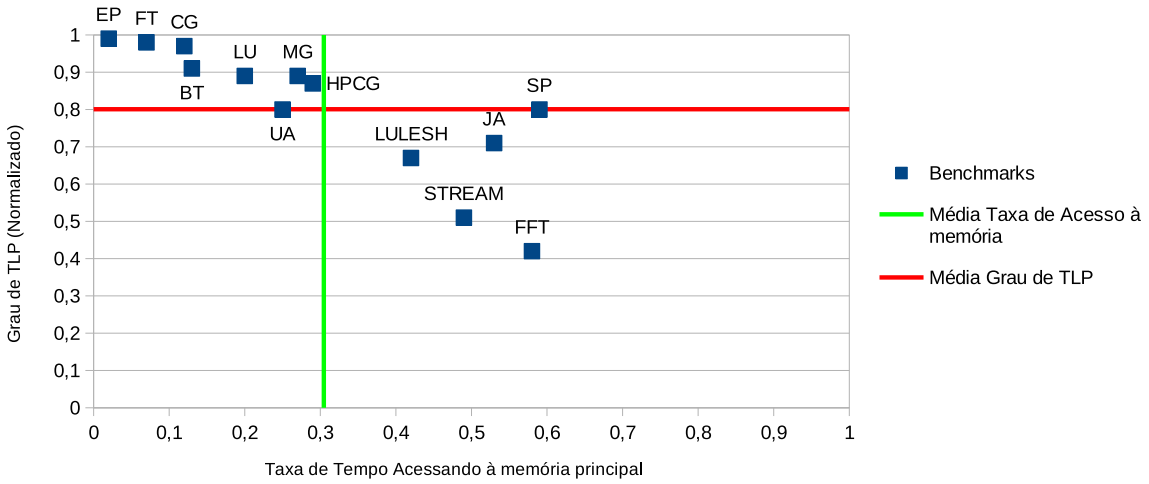
A partir dos dados coletados, as aplicações foram classificadas em duas categorias: alta taxa de comunicação e baixa taxa de comunicação. Foram considerados os valores médios das métricas utilizadas para fazer a classificação, ou seja, médias do grau de TLP

e da taxa de acerto da cache L3. A Figura 9 apresenta o gráfico dos dados coletados, bem como as médias do grau de TLP e da taxa de acesso à memória principal. As categorias definidas são:

1. ***Alta Taxa de Comunicação***: esta classe compreende as aplicações com taxas de comunicação altas, ou seja, uso da memória acima da média das aplicações, e com grau de TLP abaixo da média das aplicações. Logo, as aplicações classificadas nesta categoria são: FFT, JA, LULESH, SP e ST.

2. ***Baixa Taxa de Comunicação***: esta classe compreende as aplicações com baixa taxa de comunicação, ou seja, com grau de TLP acima da média e com tempo de acesso à memória abaixo da média. Foram classificadas nesta categoria as aplicações: BT, CG, EP, FT, HPCG, MG, LU e UA.

Figura 9 – Classificação das aplicações dos *benchmarks*



Fonte: O autor, 2021

4.1.3 Variação do TLP

Os *benchmarks* foram executados primeiramente utilizando apenas uma *thread* para avaliar a versão sequencial, sem paralelismo de instruções. Para realizar a avaliação do paralelismo serão usados número pares de *threads* para a execução dos *benchmarks* já que as arquiteturas utilizadas escalam o número de PUs em múltiplos de dois. Assim, a variação do número de *threads* é de inicialmente uma *threads*, seguido pelo uso de duas *threads* e aumentando em duas *threads* até o número total de PUs do sistema.

4.1.4 Emprego do Placement e Afinidade

As configurações de *placement* e afinidade utilizadas nas arquiteturas *multicore* estão descritas na Tabela 3. Um ponto a destacar é a arquitetura AMD Ryzen 7 que não foi avaliada com a política de *placement Sockets* por possuir apenas um processador. Caso esta política seja utilizada em sistemas com apenas um processador o comportamento é semelhante a política de afinidade *False* pois todas as PUs farão parte de um único *place*.

Tabela 3 – Configurações de *placement* e afinidade utilizadas por arquitetura

Configuração	AMD Ryzen 7 1700	Intel Xeon E5-2640	IBM Power9 8335-GTH
Placement	*Cores* e *Threads*	*Sockets*, *Cores* e *Threads*	
Afinidade	*Close* (C), *False* (F), *Master* (M), *Spread* (S), *True* (T)		

Fonte: O autor, 2021

4.1.5 Monitoramento da Temperatura

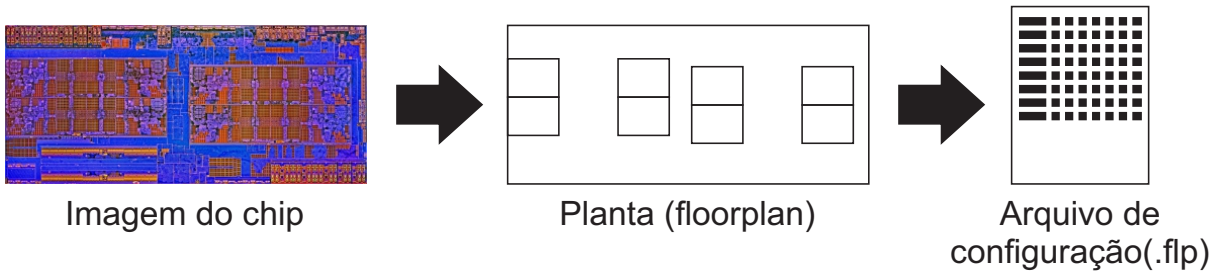
Nos sistemas com processadores Intel ou IBM a temperatura e a tensão de operação foram obtidas através do pacote do Linux *lm-sensors* pelo comando *sensors*, sendo possível coletar dados de cada *core* separadamente.

Nos sistemas com processadores AMD não é possível coletar dados por *core* devido à arquitetura não prover uma interface de leitura dos dados térmicos. Devido a isso foi utilizada uma ferramenta, o MatEx (PAGANI et al., 2015), para calcular a temperatura de cada *core* separadamente. A ferramenta calcula a temperatura de cada *core* baseado na planta da arquitetura do processador (*floorplan*) e nas leituras de potência obtidas com a biblioteca APM, que permite a leitura diretamente dos registradores do Linux.

A Figura 10 representa a forma de obtenção dos dados das arquiteturas AMD para geração do arquivo de configuração com as posições dos *cores* utilizado pelo MatEx no cálculo da temperatura. A partir da imagem do *chip* e de informações da dimensão real do dispositivo, calcula-se a escala a ser utilizada para inferir as posições dos *cores*. Após, identifica-se a posição dos *cores* no *chip* e, com base na escala, estima-se a posição real, gerando o arquivo de configuração do *floorplan* com os dados das localizações dos cores.

Em todos os cenários testados, os dados foram coletados com intervalo de um segundo durante a execução de cada aplicação. A frequência dos processadores de cada arquitetura foi configurada para a frequência de operação base de cada uma, conforme apresentado na Tabela 2, para evitar a influência de outros mecanismos de ajuste do ambiente de execução e manter a base de comparação entre as configurações testadas.

Figura 10 – Geração do arquivo de configuração da arquitetura para o MatEx



Fonte: O autor, 2021

4.1.6 Cálculo do aging

Conforme descrito na Seção 2.2, no cálculo do *aging* foi utilizada a Equação 2.1, utilizando os valores das constantes definidos nos trabalhos de Bhardwaj et al. (2006), Oboril e Tahoori (2012) e Amrouch et al. (2014), descritos na Tabela 4. Sendo que A_N é obtido conforme descrito por Amrouch et al. (2014).

Tabela 4 – Parâmetros constantes utilizados

Constante	Valor
E_0	$0,335$
n	$1/6$
E_a	$0,49eV$
ξ_1	$0,9$
ξ_2	$0,5$
ξ_3	$1,0$
ξ_4	10^{-8}
k	$8,617x10^{-5}eV/K$

Fonte: O autor, 2021

Assim as Equações 2.1, 2.2, 2.3 e 2.4 ficam na forma:

$$\Delta V_{th} \leq \int_0^1 A_N \cdot u(V_{dd}) \cdot \frac{(v(T_C) \cdot \delta_C \cdot \delta_e \cdot t_m)^{1/6}}{w(\delta_C \cdot \delta_e, T, t)^{1/3}} d\delta_e; \tag{4.1}$$

$$u(V_{dd}) = (V_{dd} - V_{th}) \cdot exp\left(\frac{V_{dd} - V_{th}}{0,335}\right) \tag{4.2}$$

$$v(T_C) = 10^{-8} \cdot exp\left(\frac{-0,49}{8,617x10^{-5}T_C}\right) \tag{4.3}$$

$$w = 1 - \left(1 - \frac{0,9 + \sqrt{1 \cdot v(T_C) \cdot (1 - \delta(t)) \cdot t_m}}{0,5 + \sqrt{v(T_C) \cdot t}}\right)^3 \tag{4.4}$$

Os dados complementares são obtidos diretamente do *hardware* a cada segundo, ou seja, $t_m = 1s$. V_{dd} é obtido usando o pacote *lm-sensors*. A temperatura (T_C) é obtida conforme descrito na seção anterior (4.1.5).

Para calcular o ciclo de trabalho de cada core (δ_C) ao executar uma aplicação, utiliza-se a porção do tempo (ciclos) total de execução em que o core esteve sob *stress*. A Equação 4.5 mostra a forma de cálculo de δ_C.

$$\delta_C = \frac{ciclos_{stress}}{ciclos_{total}} \tag{4.5}$$

Os resultados apresentados na próxima seção consideram a NBTI no pior caso. Isto é, o maior valor apresentado entre todos os *cores*.

4.2 Resultados Experimentais

Nesta seção, apresentamos a exploração de espaço e projeto com relação a NBTI e desempenho resultante da execução das aplicações descritas na Seção 4.1 com diferentes números de *threads*, políticas de *placement* e afinidade de *threads* em diferentes arquiteturas *multicore*. O objetivo principal desta seção é mostrar que não existe uma única configuração ideal de número de *threads*, *placement* e afinidade que apresenta os melhores resultados de NBTI para todas as aplicações e arquiteturas, destacando a motivação desta proposta de dissertação.

As Figuras 11 e 14 mostram os resultados de NBTI (barras) e tempo de execução (círculo) para as duas classes de *benchmarks* considerando a média geométrica de todas as aplicações em cada classe. Cada barra mostra o NBTI médio de todos os *cores* em cada *socket*. Quanto menor o valor, melhor. Cada barra representa os resultados de uma configuração no *eixo x* dado por: o número de *threads* usados no experimento (por exemplo, 2 *threads*, 4 *threads* e assim por diante); a política de *placement* (*Cores*, *Sockets* e *Threads*); e a estratégia de afinidade de *thread* empregada, onde C significa *close*, F - *false*, M - *master*, S - *spread*, e T - *true*. Para esta discussão, nos referimos à política de *placement* e afinidade de *threads* como um par. Por exemplo, com *Cores-Close* nos referimos ao uso de *Cores* como *placement* e *Close* para a afinidade. Como os sistemas Intel Xeon 32-core e IBM Power9 possuem dois *sockets*, separamos os resultados de cada *socket* em barras coloridas distintas. O melhor resultado da NBTI para cada *socket* está destacado na cor verde e o menor tempo está representado com um asterisco. Esses resultados serão apresentados nas próximas seções. A seguir, discutimos os resultados da NBTI de cada classe. Adicionalmente, os resultados de cada aplicação em cada processador podem ser encontrados no Apêndice A.

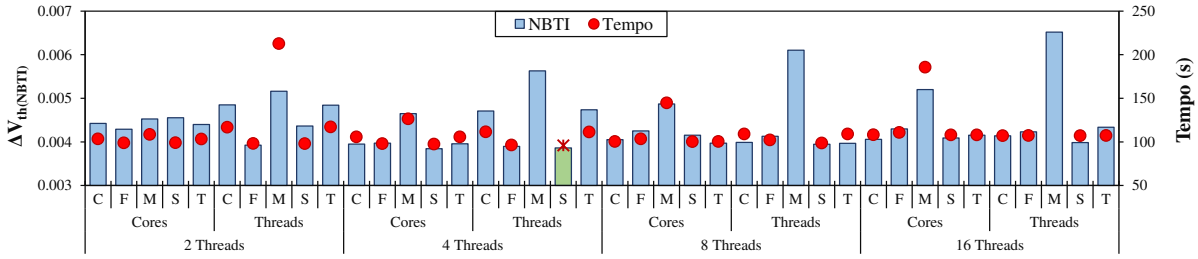
4.2.1 Aplicações com Alta Taxa de Comunicação

Ao considerar a média geométrica de todas as aplicações com alta taxa de comunicação, o melhor resultado de NBTI no sistema AMD de 16-core (Figura 11a) foi alcançado com a configuração *Threads-Spread* rodando com apenas quatro *threads*. Neste caso, o impacto da NBTI no envelhecimento do processador é 10% menor do que o obtido pela execução padrão de aplicações paralelas (o número máximo de *threads* de hardware com o par *Threads-False*). No entanto, quando o sistema *multicore* é composto de mais de um *socket*, as políticas de *placement* e afinidade afetam cada *socket* de maneira diferente. Por exemplo, para o sistema Intel 32-core (Figura 11b), a configuração *Cores-True* com oito *threads* apresenta o melhor resultado de NBTI para o *socket-0* enquanto que a configuração *Threads-Spread* com quatro *threads* é melhor para o *socket-1*. Por fim, para o sistema IBM Power9 (Figura 11c), a execução da configuração *Cores-Close* com 40 *threads* apresentou o melhor resultado de NBTI para ambos *sockets*. Adicionalmente, como pode ser observado, a configuração que menos degrada o processador devido a NBTI não é a mesma que oferece o melhor tempo de execução, na grande maioria dos casos.

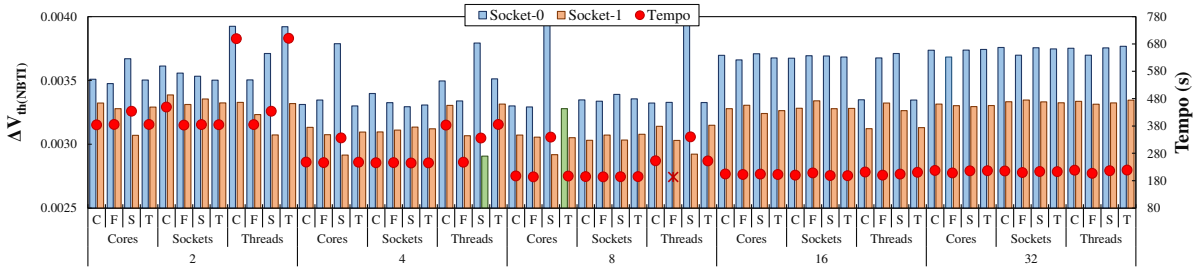
Para melhor entender o amplo espaço de otimização no uso de distintas estratégias de *placement* e afinidade de *threads* na NBTI, a Tabela 5 mostra a melhor configuração para cada aplicação em cada sistema *multicore*. Conforme pode-se observar, existe uma variabilidade significativa na configuração que atinge o melhor resultado de NBTI. Por exemplo: a aplicação FFT possui diferentes graus de TLP para as arquiteturas, variando de oito a 160 *threads*; ainda apresenta variabilidade da política de *placement* entre *Cores* e *Threads*; e, a estratégia de afinidade varia em *True*, *False* ou *Spread*. Portanto, escolher uma única configuração para executar a mesma aplicação em diferentes arquiteturas pode não levar ao melhor resultado de NBTI.

Com o objetivo de observar o impacto do uso de distintas estratégias de alocação no envelhecimento do processador, analisa-se a Figura 12. Ela mostra o comportamento de diferentes estratégias de *placement* e afinidade em execução com oito *threads* no sistema Intel de 32-core. Quando considera-se a configuração *Cores-False*, porque as *threads* não criam pontos de calor conforme são distribuídas pelos *sockets*, o impacto da NBTI no envelhecimento de todos os *cores* é homogêneo (desvio padrão de apenas 3,25%). Nesse caso, as *threads* são movidas entre as PUs do sistema. Por outro lado, para o par *Cores-Spread*, os *cores* dos diferentes *sockets* envelhecem de forma mais heterogênea, já que todas as *threads* são alocadas no *Socket-0*. Existem também cenários onde pares de estratégias de posicionamento e afinidade têm resultados semelhantes considerando a média geométrica de todos os *cores* (Figura 11), mas afetam o envelhecimento de cada *core* de maneira distinta. A Figura 13 descreve esse comportamento para a execução de duas estratégias diferentes com quatro *threads* no processador AMD. Embora envelheçam todo o processador igualmente (Figura 11), quando o par *Close-Spread* é definido, os núcleos 0 e 6 envelhecerão mais rápido porque as *threads* são alocadas para cada um deles

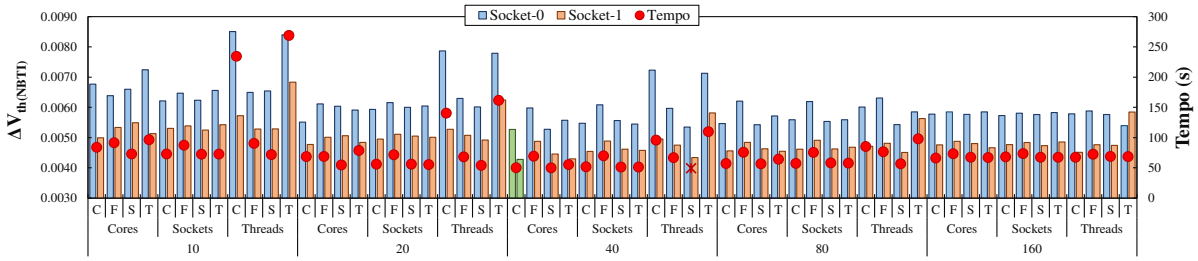
Figura 11 – Resultados das aplicações com Alta Taxa de Comunicação



(a) AMD Ryzen 7 16-core



(b) Intel Xeon 32-core



(c) IBM Power9

Fonte: O autor, 2021

durante toda a execução. Por outro lado, quando o par *Threads-False* é usado, as *threads* não são limitadas à um *core* específico, reduzindo a possibilidade de criação de pontos de calor.

Além disso, ao considerar todos os resultados obtidos na Figura 11, as seguintes observações podem ser destacadas:

- Quanto maior o número de *threads*, mais homogêneo é o impacto do NBTI no envelhecimento de todos os *cores*. Ou seja, cada *core* tende a envelhecer na mesma proporção.

- Por outro lado, quanto menor o número de *threads*, mais heterogêneo é o impacto da NBTI. Ou seja, alguns *cores* envelhecerão mais rápido do que outros.

- Em arquiteturas de multiprocessadores (Intel 32 *cores* e IBM Power9 160 *cores*), o impacto da NBTI no envelhecimento é mais significativo no *Socket-0* sendo 10,9 % e 21,9 % vezes maior em média, respectivamente. Isso se deve à *thread* mestre: uma

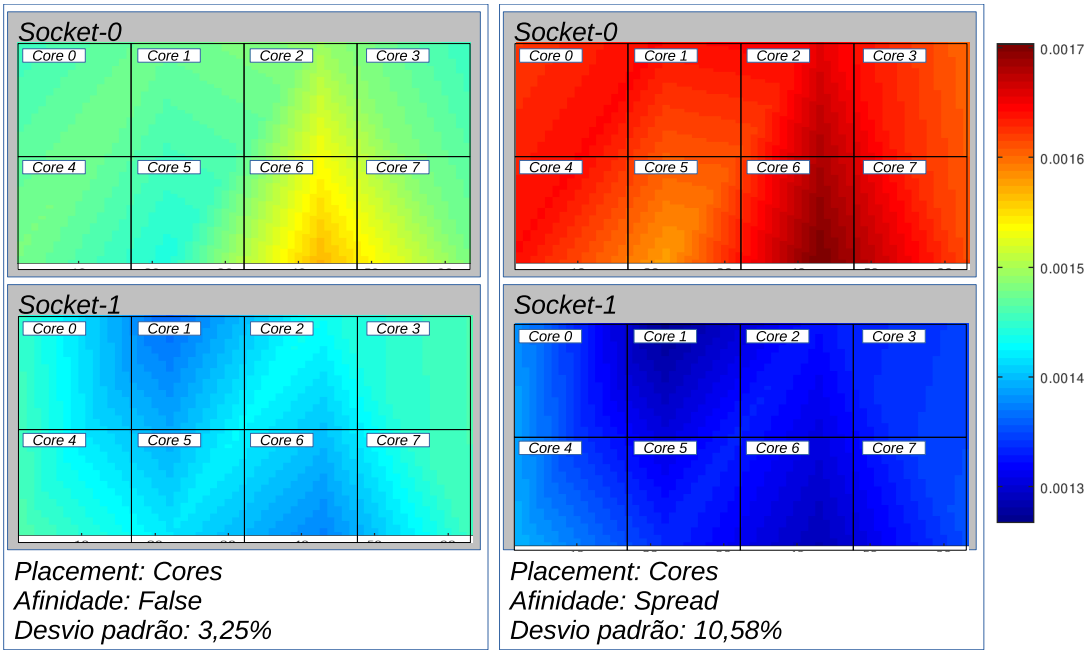
Tabela 5 – Melhores configurações por arquitetura

	Aplicação	AMD Ryzen 7 16-cores	Intel Xeon 32-cores		IBM Power9 40-cores	
			Socket-0	*Socket-1*	*Socket-0*	*Socket-1*
Alta Com.	FFT	16 - C - Tr	32 - C - F	8 - C - Sp	160 - T - F	40 - C - Tr
	JA	4 - T - F	8 - C - Tr	8 - T - F	160 - T - Tr	40 - S - Cl
	LULESH	8 - C - M	8 - C - Tr	8 - C - Sp	40 - C - Cl	80 - T - Cl
	SP	4 - C - F	8 - C - Tr	4 - T - Sp	40 - T - Sp	40 - C - Tr
	ST	16 - C - Tr	4 - C - Sp	8 - S - Sp	160 - T - Tr	40 - C - Cl
Baixa Comunic.	HPCG	2 - C - F	8 - S - Cl	8 - C - Sp	160 - T - Tr	160 - T - Cl
	BT	8 - C - Sp	16 - T - Tr	8 - C - Sp	160 - T - Tr	160 - T - Cl
	CG	4 - T - Sp	32 - S - Cl	8 - T - Sp	160 - T - Tr	160 - C - Tr
	FT	8 - T - F	32 - S - F	16 - T - Sp	40 - C - Cl	40 - C - Cl
	EP	16 - C - Tr	32 - T - Sp	32 - S - F	160 - T - Tr	160 - T - Cl
	LU	2 - T - F	32 - C - Tr	32 - C - Tr	80 - T - Tr	160 - T - Cl
	MG	2 - C - F	16 - T - Tr	16 - T - Cl	40 - C - Sp	40 - C - Cl
	UA	16 - T - Cl	16 - T - Cl	8 - C - Sp	160 - T - Tr	160 - T - Cl

Formato da configuração: Número de *threads* - *Placement* - Afinidade. Políticas de *placement*: (**S**)ockets; (**C**)ores; e, (**T**)hreads. Estratégias de afinidade: (**Cl**)ose; (**F**)alse; (**M**)aster; (**Sp**)read; e, (**Tr**)ue.

Fonte: O autor, 2021

Figura 12 – Média da NBTI por *core* da execução de uma aplicação com alta taxa de comunicação no Intel Xeon 32-core



Fonte: O autor, 2021

vez que executa partes sequenciais de cada aplicação, é sempre colocado em uma PU do *Socket-0*.

- Independentemente do sistema *multicore*, os piores resultados são quando a afinidade de *thread* é definida como *master*. Isso ocorre porque a alocação de cada nova *thread* está no mesmo lugar que a *thread* mestre. Portanto, as *threads* competirão por recursos compartilhados, gastando mais tempo para computar a aplicação e criando pontos de calor devido à maior dissipação de energia do local afetado. Devido a este comportamento, tais resultados foram omitidos dos gráficos dos processadores Intel e IBM, uma vez que o valor de NBTI e tempo de execução é extremamente alto quando comparado às demais políticas.

Esta subseção mostrou que, para aplicativos com alta demanda de comunicação, o número de *threads* e estratégias de *placement* e afinidade devem ser considerados juntos quando se trata de encontrar a melhor configuração para otimizar o *aging*.

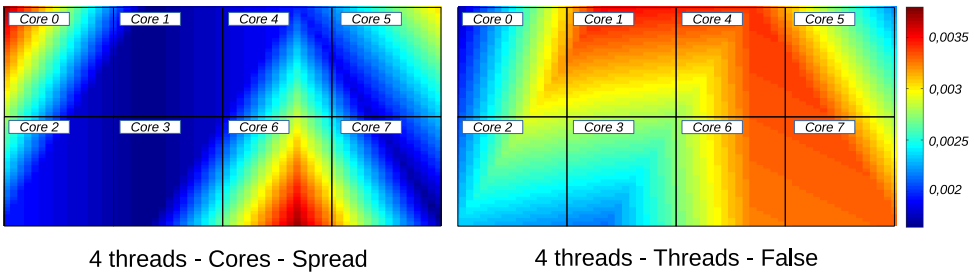
4.2.2 Aplicações com Baixa Taxa de Comunicação

Ao contrário da classe anterior, a taxa de envelhecimento desta é mais influenciada pelo número de *threads* do que pelas estratégias de posicionamento e afinidade (Figura 14). Isso se explica pelo alto grau de exploração do TLP e pela baixa taxa de comunicação, o que reduz a competição por recursos compartilhados.

Ao considerar a média geométrica de todo o conjunto de *benchmarks*, o melhor resultado no sistema AMD 16-core foi alcançado com as seguintes configurações: *Threads* como política de posicionamento com estratégias de afinidade *close*, *false*, *spread* ou *true* rodando com 16 *threads*. Para o sistema Intel 32-core, a política de posicionamento *Sockets* com a estratégia de afinidade *close*, *spread* ou *true* rodando com 32 *threads* entregou os melhores resultados. Nesse caso, o NBTI foi 3 % menor do que a execução padrão. Finalmente, para o sistema IBM Power9 de 160 *cores*, a configuração *Threads-True* executando 160 *threads* apresentou melhores resultados. Nesse caso, o NBTI foi 24 % menor do que a combinação padrão.

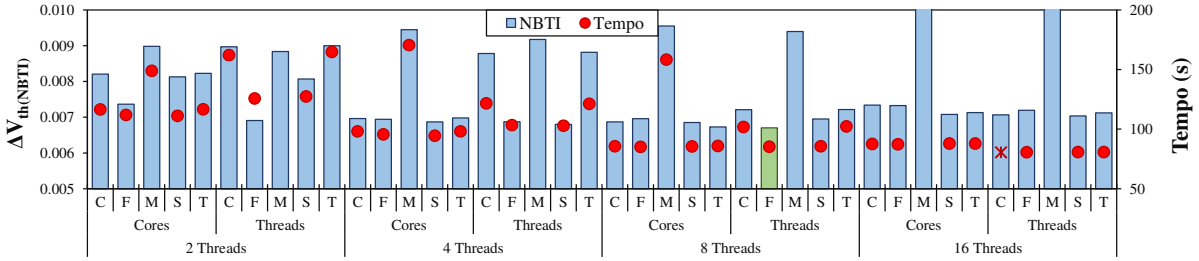
Com base nos resultados obtidos em cada sistema, podemos destacar as seguintes observações:

Figura 13 – NBTI - Execução no AMD 16-core com quatro *threads*

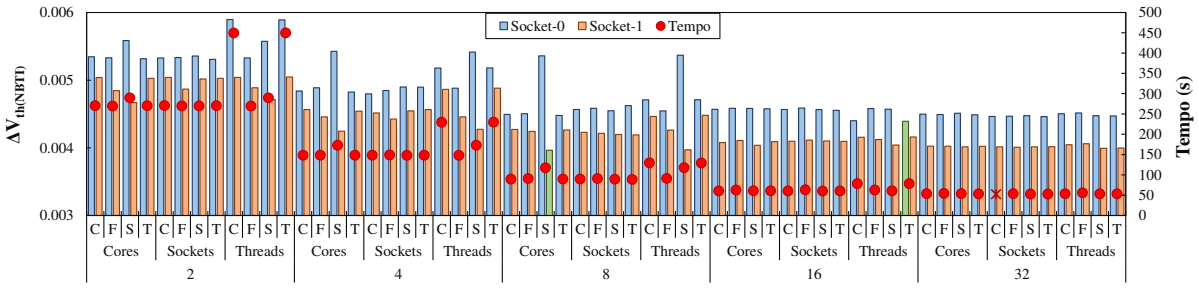


Fonte: O autor, 2021

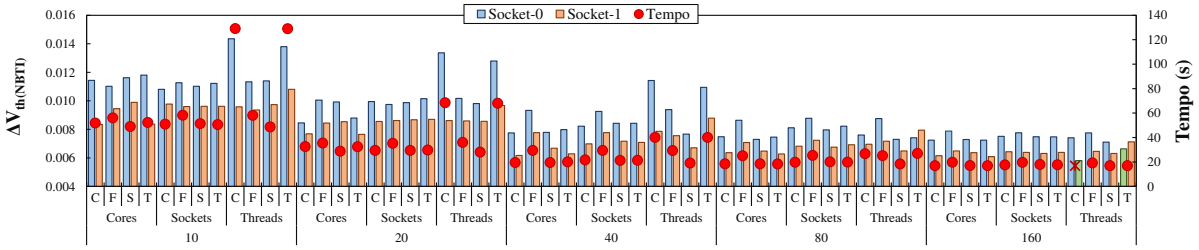
Figura 14 – Resultados das aplicações com Baixa Taxa de Comunicação



(a) AMD Ryzen 7 16-core



(b) Intel Xeon 32-core



(c) IBM Power9

Fonte: O autor, 2021

- Quanto maior o número de *threads* ativas, pior é o comportamento da configuração padrão (*Threads-False*) devido à grande quantidade de troca de contexto, uma vez que as *threads* não são limitados a um local específico. Portanto, as execuções com o número máximo de *threads* entregaram o pior resultado para tal configuração, como se pode observar no sistema IBM Power9 rodando com 160 *threads*.

- As configurações com afinidade *master* também tiveram os piores resultados devido aos mesmos motivos discutidos na Seção 4.2.1.

5 ABORDAGEM

No capítulo anterior, mostrou-se, através da análise da NBTI, que variar o grau de TLP e a política de alocação de *threads* impacta no *aging* de diferentes formas. Embora a análise permita inferir o comportamento da NBTI conforme o tipo de aplicação, foi constatado que não existe uma política ideal para todas as aplicações. Portanto, este capítulo apresenta uma metodologia de redução do *aging* através da variação do grau de TLP e da definição de políticas de alocação de *threads* compatíveis com a IPP OpenMP.

A metodologia proposta utiliza as políticas de *placement* e de afinidade disponibilizadas pelo OpenMP juntamente com a variação do TLP. A essa metodologia foi atribuída o nome de *Aging Aware Thread Scheduling* (em português, Escalonamento de Threads com Envelhecimento Consciente) (MEDEIROS et al., 2020) sendo simplesmente identificada pela sigla **AATS**. Para tanto, propomos duas abordagens: uma *offline*, que atua antes da execução da aplicação, e uma *online*, que ajusta os parâmetros em tempo de execução. Ambas utilizam o mesmo algoritmo de aprendizagem na exploração do espaço de soluções. O restante do capítulo está organizado da seguinte forma: a estratégia *offline* é apresentada na Seção 5.1, descrevendo o emprego do algoritmo de aprendizagem; e, de forma semelhante, a abordagem *online* é descrita na Seção 5.2.

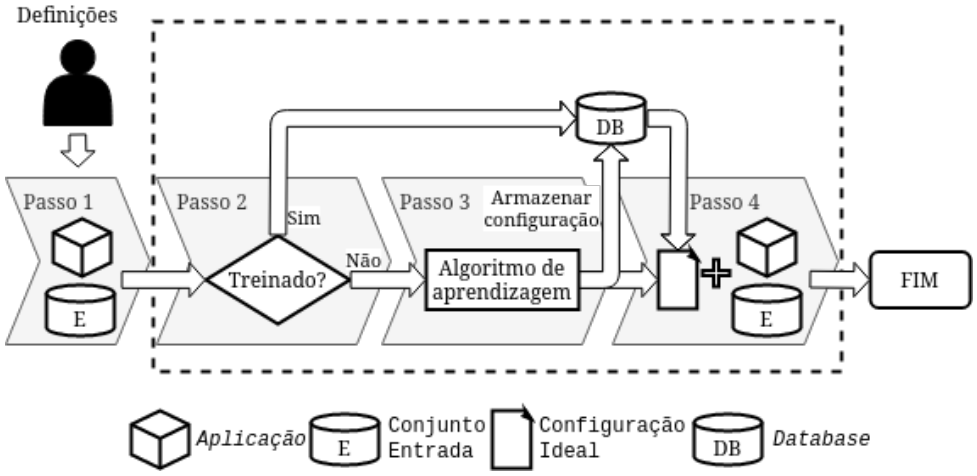
5.1 AATS Offline

Considerando a aplicação de um modo mais amplo, sem detalhar as peculiaridades de sua execução, foi utilizada uma abordagem aplicada ao contexto externo à execução da aplicação. Dessa forma, a avaliação da configuração utilizada depende dos dados globais da execução. Essa abordagem é denominada como o **AATS *offline***, já que não é aplicada durante a execução da aplicação e sim realiza o monitoramento para posteriormente avaliar a configuração utilizada.

Sejam os recursos de *hardware* e as estratégias de alocação descritos como: o conjunto de c distintas unidades de processamento (PUs, ou seja, *threads* de *hardware* ou *cores* lógicos) definido por $C = \{C_1, C_2, ..., C_c\}$; o conjunto de p distintas políticas de *placement* definido por $P = \{P_1, P_2, ..., P_p\}$; e, t diferentes estratégias de afinidade de *threads* definidas por $T = \{T_1, T_2, ..., T_t\}$. O problema objeto de estudo é encontrar um subconjunto de *threads*/*cores* em C, com uma política de *placement* em P, e com a afinidade de *threads* em T que otimiza o *trade-off* entre desempenho e *aging* do processador na execução de uma aplicação paralela A. A estratégia é apresentada na Figura 15 e é definida como uma sequência de passos, conforme descrito a seguir.

Primeiramente, o usuário define a aplicação e sua entrada (Passo 1). Importante destacar que o número de interações de uma aplicação pode depender da entrada, assim a definição da configuração ideal dependerá do conjunto de entrada. No próximo passo é verificado se o algoritmo já fez o processo de aprendizagem para o conjunto aplicação

Figura 15 – AATS *offline*



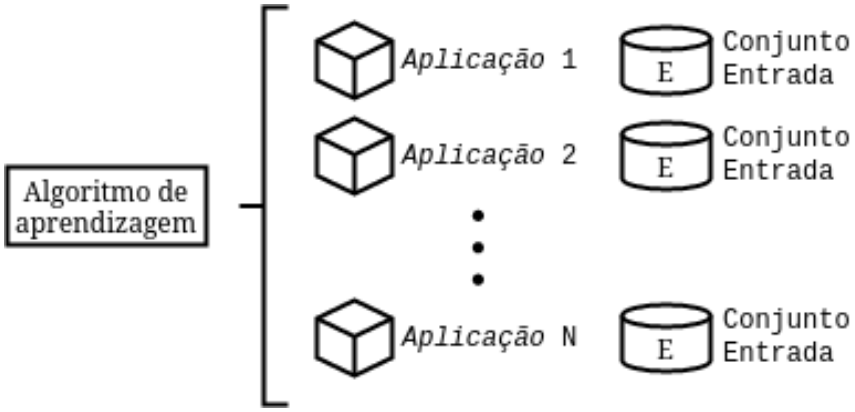
Fonte: O autor, 2021

e entradas. Essa verificação busca em uma base de dados se há alguma configuração já avaliada para a aplicação e a entrada definida (Passo 2). A existência de uma configuração caracteriza que o processo de aprendizagem foi executado em algum momento. Para controlar a situação é armazenado o estado em que se encontra o processo de busca. Quando a busca é concluída, define-se como estado final para indicar que a configuração foi determinada como a ideal. Caso o processo de aprendizagem já tenha sido realizado (estado final), simplesmente executa-se a aplicação com a configuração armazenada no banco de dados (Passo 4). Assim, a configuração ideal é conhecida e o ambiente é configurado para a aplicação ser executada. No caso em que ainda não houve treinamento ou que o treinamento ainda não tenha sido concluído, o processo de busca da configuração ideal é realizado seguindo o algoritmo de aprendizado (Passo 3), que será descrito a seguir. Logo, para manter o controle do processo de busca é utilizada a base de dados (DB) para armazenar as informações das execuções anteriores. Adicionalmente, a Figura 16 apresenta o controle da execução da **AATS *offline***, que é feito através do par aplicação-conjunto de entrada, cujas configurações e estado estão armazenados em um banco de dados.

O algoritmo de aprendizagem realiza a busca da configuração ideal conforme os valores obtidos da execução. Denota-se na Equação 5.1 a função $\mathbb{M}$ como a medida obtida da execução de uma aplicação paralela cujo domínio é o espaço de busca compreendido pela combinação dos distintos *cores* de C aplicando as políticas de *placement* de P e estratégias de afinidade de *threads* de T.

Definindo como ϕ o conjunto de configuração de c, p e t e ϕ' como o valor da métrica avaliada para ϕ na qual busca-se aproximar do máximo valor de $\mathbb{M}$. Feitas essas considerações, a estratégia de busca pela solução ideal é descrita no pseudocódigo do Algoritmo 1. Na figura 17 é representada a sequência de evolução do algoritmo até chegar

Figura 16 – Controle do AATS *offline*



Fonte: O autor, 2021

na solução ideal.

$$\mathbb{M} : (C \times P \times T) \rightarrow \mathbb{R}^{+} \tag{5.1}$$

Primeiramente são caracterizadas as entradas necessárias, sendo elas: o conjunto de *cores* C; o conjunto de políticas de *placement* P; o conjunto de estratégias de afinidade de *threads* T para uma aplicação A; e, dois parâmetros. Estes parâmetros são o número de *threads* inicial (α) para executar a aplicação e o fator de incremento do número de *threads* (β). O parâmetro α é calculado e definido pelo algoritmo de busca de acordo com a arquitetura alvo. O algoritmo então seleciona a política de *placement* padrão e a estratégia de afinidade de *threads* para a fase inicial. O **AATS *offline*** utilizada possui três fases distintas de otimização. Na primeira fase é realizado o ajuste do número de *threads*, na segunda o ajuste da politica de *placement* e por fim, o ajuste da estratégia de afinidade de *threads*.

No ajuste do número de *threads* é empregado o algoritmo *hill-climbing* modificado. Inicialmente, a aplicação é executada com α *threads* e é medido o M'. Para as próximas execuções o número de *threads* é definido utilizando β como fator multiplicativo de α, sendo o novo valor de $\alpha = \alpha * \beta$. Porém, o valor de α somente é atualizado com base no fator β enquanto se maximiza a função de avaliação $M' = \mathbb{M}(\phi)$ (linhas 6 a 9). Neste momento o *hill-climbing* utiliza um passo numa escala geométrica com razão β. Uma vez que não é encontrado um valor maior que o anterior entende-se que há um máximo local. Então o algoritmo continua a buscar uma solução ϕ' dentro do intervalo entre o máximo valor encontrado e o último ponto analisado. Esse intervalo é delimitado na linha 11 e o ponto médio é definido e avaliada a função M', assim o algoritmo *hill-climbing* tem como passo a metade dos pontos a avaliar entre o mínimo e o máximo podendo aumentar ou diminuir o número de *threads* a avaliar. Avaliado o ponto médio define-se o próximo subintervalo a ser analisado, à direita caso M' apresente um valor maior ou à esquerda caso contrário. Uma vez que a busca pelo número ideal de *threads*

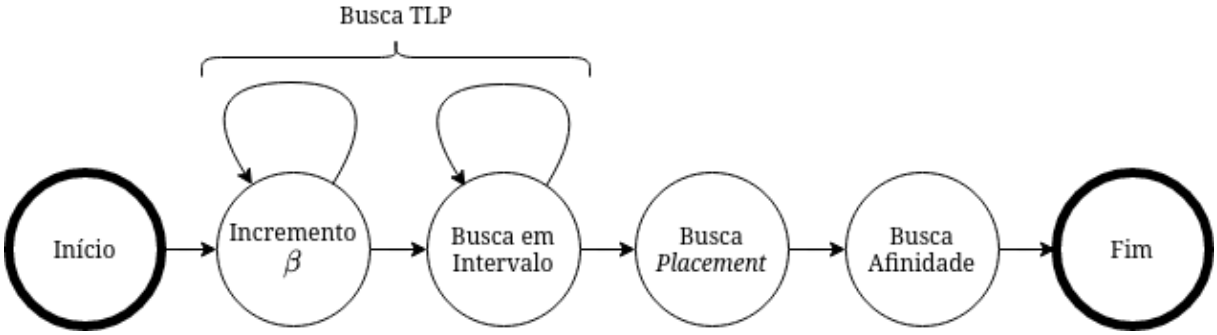
Algorithm 1 Algoritmo de aprendizagem

Input: $C \leftarrow \{C_1, C_2, \ldots, C_k\}$: conjunto de threads/cores
$P \leftarrow \{HWThreads, Cores, Sockets\}$: conjunto de políticas de placement
$T \leftarrow \{false, close, master, spread, true\}$: conjunto de afinidades de *threads*
α: número inicial de *threads*
β: fator de incremento do número de *threads*

1: $p \leftarrow P\{1\}$: define a primeira política de *placement* a ser avaliada
2: $t \leftarrow T\{1\}$: define a primeira estratégia de afinidade a ser avaliada
3: $\phi(p,t,\alpha) \leftarrow -\infty$: melhor combinação de *placement*, afinidade e grau de TLP encontrado até o momento
4: $\phi' \leftarrow -\infty$: valor máximo encontrado até o momento
5: $M' \leftarrow \mathbb{M}(p,t,\alpha)$
6: **while** $M' \geq \phi'$ and $\alpha \leq totalCores$ **do**
7: $\phi' \leftarrow M'$ and $\phi \leftarrow (p,t,\alpha)$
8: $\alpha \leftarrow \alpha \times \beta$ and $M' \leftarrow \mathbb{M}(p,t,\alpha)$
9: **end while**
10: **if** $M' \leq \phi'$ and $\alpha \leq totalCores$ **then**
11: $upper \leftarrow \alpha$ and $lower \leftarrow \alpha/\beta$
12: **while** $lower \leq upper$ **do**
13: $\alpha' \leftarrow (upper + lower)/2$
14: $M' \leftarrow \mathbb{M}(p,t,\alpha')$
15: **if** $M' \geq \phi'$ **then**
16: $\phi' \leftarrow M'$ and $\phi \leftarrow (p,t,\alpha')$
17: $lower \leftarrow \alpha'$
18: **end if**
19: **if** $M' < \phi'$ **then**
20: $upper \leftarrow \alpha'$
21: **end if**
22: **end while**
23: **end if**
24: **for** each p in P **do**
25: $M' \leftarrow \mathbb{M}(p,t,\alpha)$
26: **if** $M' \geq \phi'$ **then**
27: $\phi' \leftarrow M'$ and $\phi \leftarrow (p,t,\alpha)$
28: **end if**
29: **end for**
30: **for** each t in T **do**
31: $M' \leftarrow \mathbb{M}(p,t,\alpha)$
32: **if** $M' \geq \phi'$ **then**
33: $\phi' \leftarrow M'$ and $\phi \leftarrow (p,t,\alpha)$
34: **end if**
35: **end for**
36: **return** $\phi(p,t,\alpha)$

foi concluída, é feita a variação das políticas de *placement* (linhas 24 a 29). Da mesma forma que na etapa anterior, busca-se maximizar a função M' definindo a política p a ser utilizada. As estratégias de afinidade de *threads* são avaliadas por último, após a definição da política de *placement* ideal. Finalmente, após o processo de busca é definido o conjunto de configurações ϕ a ser utilizado para a aplicação.

Figura 17 – Controle de estados



Fonte: O autor, 2021

5.2 AATS Online

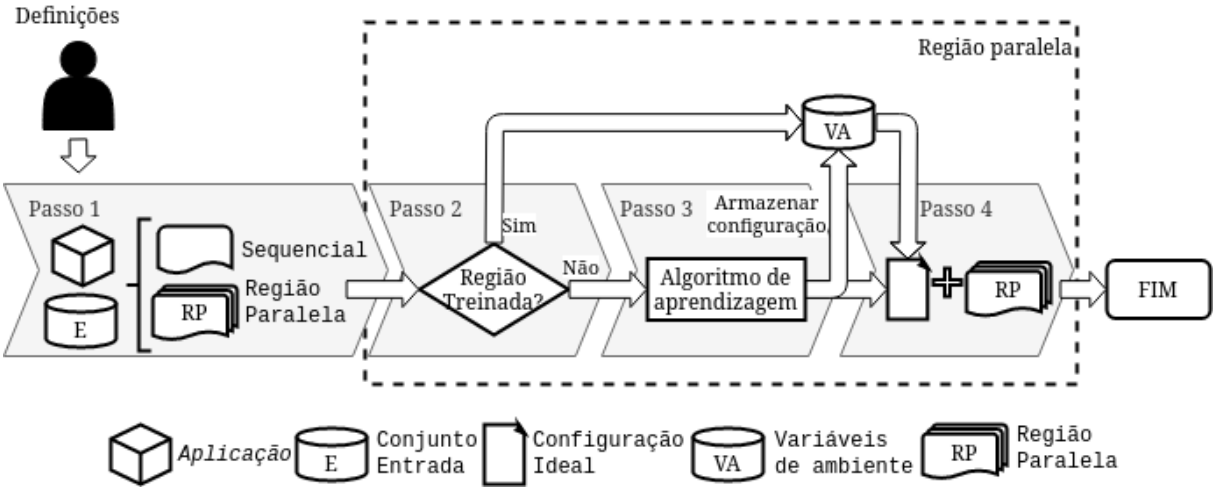
Sejam os recursos de *hardware* e as estratégias de alocação descritos como: o conjunto de c distintas unidades de processamento (PUs, ou seja, *threads* de *hardware* ou *cores* lógicos) definido por $C = \{C_1, C_2, ..., C_c\}$; o conjunto de p distintas políticas de *placement* definido por $P = \{P_1, P_2, ..., P_p\}$; t diferentes estratégias de afinidade de *threads* definidas por $T = \{T_1, T_2, ..., T_t\}$; e, o conjunto de r regiões paralelas da aplicação definido por $R = \{R_1, R_2, ..., R_r\}$

O problema objeto de estudo na abordagem *online* é encontrar um subconjunto de *threads*/*cores* em C, com uma política de *placement* em P, e com a afinidade de *threads* em T que reduz o *trade-off* entre desempenho e *aging* do processador na execução de uma região paralela de R pertencente à aplicação A.

Diferente do **AATS *offline*** apresentada anteriormente, na versão **AATS *online*** o algoritmo apresentado é aplicado durante a execução da aplicação. Assim, enquanto a aplicação é executada são realizados ajustes no TLP, *placement* e afinidade. No processo de otimização é considerada cada região paralela. Deste modo o processo de otimização segue de forma diferente em cada região. Na Figura 18 é apresentado o fluxo do processo de busca da solução.

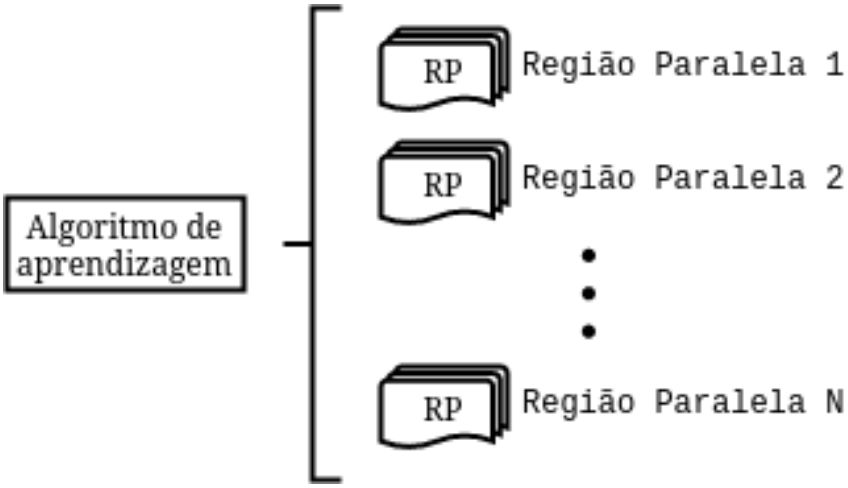
Uma aplicação paralela possui regiões sequenciais e paralelas. Devido as regiões paralelas possuírem características diferentes, a métrica utilizada no algoritmo de aprendizagem apresentado poderá fornecer valores distintos, pois cada região paralela poderá possuir um conjunto de instruções distinto. Quando o OpenMP encontra uma região paralela utiliza-se o algoritmo para verificar se essa região já foi treinada (Passo 2). Esta verificação é feita em variáveis de ambiente que controlam o processo de aprendizagem. Caso a região ainda não tenha definida uma configuração ideal, o processo de aprendizagem é executado seguindo o algoritmo apresentado na seção anterior (Passo 3). Da mesma forma que na busca *offline*, para manter o controle do processo de busca é utilizado estados para indicar a situação da busca. As configurações aplicadas são armazenadas

Figura 18 – AATS *Online*



Fonte: O autor, 2021

Figura 19 – Controle do AATS *online*



Fonte: O autor, 2021

em estruturas de dados de controle e a região é executada. Uma vez que o processo de aprendizagem está concluído para a região paralela, essa é executada com a configuração específica encontrada para ela (Passo 4).

Na Figura 19 é apresentada a forma de controle do aprendizado aplicado no **AATS *online***. Nessa abordagem, o controle utiliza estruturas de dados para controlar a busca em cada região paralela de forma isolada.

Denota-se na Equação 5.2 a métrica obtida da execução de uma aplicação paralela com r regiões paralelas, em c distintos *cores* aplicando a política de *placement* p e estratégia de afinidade de *threads* t.

$$\mathbb{M} : (R \times C \times P \times T) \rightarrow \mathbb{R}^{+} \tag{5.2}$$

Define-se como ϕ o conjunto de configuração de c, p e t para a região r com o valor da métrica armazenado em ϕ', na qual espera-se aproximar do máximo valor de $\mathbb{M}$. É utilizado o mesmo algoritmo apresentado na Seção 5.1 porém aplicado a cada região. Assim, para cada região paralela r há um processo de aprendizagem individualizado diferentemente do **AATS *offline*** em que o processo de aprendizagem é por aplicação.

6 RESULTADOS

Neste capítulo, apresenta-se os resultados obtidos na avaliação das abordagens **AATS *offline*** e **AATS *online*** descritas no Capítulo 5.

6.1 Análise da Busca Offline

6.1.1 Metodologia da Busca Offline

Na avaliação do **AATS *offline*** foram utilizados os *benchmarks* e arquiteturas descritas na Seção 4.1. Os resultado são apresentados na forma de análise do impacto no tempo de vida do processador e quanto processamento ainda será possível realizar com a redução do *aging*, ou seja, até o final da vida útil do processador. Será comparado à solução ideal, apresentada no Capítulo 4 referindo-se apenas como ***Best-DSE***, e ao tempo de aprendizagem e/ou impacto na execução. O tempo de aprendizagem compreende o tempo gasto para que o algoritmo convirja para uma solução ideal. Ainda, será comparado à forma padrão de execução do OpenMP, definida como ***OMP_STD***. O ***OMP_STD*** utiliza o número máximo de *threads* suportado pela arquitetura e alocação *Threads-False*.

6.1.2 Resultados Experimentais da Busca Offline

Nas Tabelas 6 e 7 estão apresentados os resultados da aplicação do algoritmo na execução dos *benchmarks*. É apresentada a solução encontrada para cada *benchmark* em cada arquitetura avaliada utilizando o algoritmo proposto conforme descrito na Seção 5.1, ou seja, a configuração de número de *threads*, *placement* e afinidade ideal que maximiza o *trade-off* entre desempenho e *aging*.

A diferença do *trade-off* entre desempenho e *aging* da solução para o ***Best-DSE*** está descrita na Tabela 7. Valores iguais a 1.00 significam que a solução encontrada é igual à ***Best-DSE***. O **AATS *offline*** foi capaz de encontrar a solução ideal ou uma solução próxima a essa. Considerando a média geométrica entre todos os *benchmarks* em cada arquitetura, o **AATS *offline*** ficou apenas 1,02 vezes distante da ***Best-DSE*** na arquitetura AMD. Para as arquiteturas Intel e IBM, o **AATS *offline*** ficou com 1,07 e 1,19 vezes distante da solução ideal, respectivamente. Embora a arquitetura IBM possua um grande número de *cores* a diferença para o *Best-DSE* manteve-se pequena. Esse incremento na diferença entre as soluções está relacionado à quantidade de possíveis configurações para uma arquitetura. Por exemplo, no caso da IBM há 2400 possíveis configurações, já na Intel há 480 e na AMD 240.

Ainda, na Tabela 7, está a diferença no tempo de aprendizagem para cada *benchmark* em cada arquitetura, ou seja, a taxa de aumento do tempo gasto comparado à ***Best-DSE***.Nesta análise, quanto maior o percentual maior é a sobrecarga da estratégia de busca.

Tabela 6 – Configurações encontradas pelo ATTS *offline* e DSE

Bench.	***Solução AATS offline***			***Solução Best-DSE***		
	AMD	***Intel***	***IBM***	***AMD***	***Intel***	***IBM***
FFT	8-T-Tr	8-S-F	10-C-F	16-C-Tr	2-C-F	20-C-F
HPCG	8-T-Sp	8-C-Tr	20-C-F	8-T-Sp	8-S-Cl	20-C-F
JA	4-T-Sp	2-C-F	20-S-Cl	4-T-Sp	4-C-Tr	20-S-Cl
LL	8-T-Sp	4-S-Sp	10-T-F	8-T-Sp	4-S-Sp	10-T-F
BT	8-C-Sp	8-S-F	40-S-F	8-C-Sp	16-T-Tr	80-C-Cl
CG	8-T-F	8-C-Tr	20-T-F	8-T-F	16-T-Tr	160-T-Tr
EP	16-T-Cl	32-S-F	160-C-Tr	16-T-Cl	32-S-F	160-C-Tr
FT	8-T-F	16-T-Tr	160-C-Sp	8-T-F	16-T-Tr	80-C-Cl
LU	8-T-F	8-C-Tr	80-S-Sp	16-T-Tr	16-T-Cl	40-C-Cl
MG	8-C-Tr	4-T-F	40-S-Cl	8-C-Tr	16-T-Tr	40-C-Sp
SP	4-C-Cl	8-C-Tr	20-S-Tr	4-C-Cl	8-C-Tr	20-S-Tr
UA	16-C-Sp	8-S-Tr	120-T-Tr	16-C-Sp	16-T-Cl	160-T-Tr
ST	4-T-Sp	2-C-F	10-T-F	8-T-Sp	4-S-Sp	5-T-Sp

Formato da configuração: Número de *threads* - *Placement* - Afinidade. Políticas de *placement*: (**S**)ockets; (**C**)ores; e, (**T**)hreads. Estratégias de afinidade: (**Cl**)ose; (**F**)alse; (**M**)aster; (**Sp**)read; e, (**Tr**)ue.

Fonte: O autor, 2021

Além disso, por causa do algoritmo iniciar a busca da solução pelo ajuste ideal de *threads* para executar a aplicação antes de avaliar o *placement* e a afinidade, o número de configurações avaliadas durante a busca é significantemente reduzido comparado ao realizado pela DSE. Como resultado, o **AATS *offline*** foi capaz de atingir resultados similares à *Best-DSE* executando, na média de todos os *benchmarks*, somente 1,9% das possíveis soluções da DSE na arquitetura IBM, 5,4% na arquitetura Intel e 10,4% na arquitetura AMD. Observa-se que quanto maior o espaço de exploração maior a diferença da solução encontrada comparada à ***Best-DSE***, porém com tempo inferior para achar uma solução comparada à DSE.

A Tabela 8 apresenta a diferença percentual do *trade-off* entre desempenho e *aging* da solução para o ***Best-DSE*** e para a configuração padrão do OpenMP (***OMP_STD***). Valores iguais a 0.00 significam que a solução encontrada é igual à melhor configuração encontrada na exploração do espaço de projeto (DSE). Valores negativos mostram que a solução encontrada apresenta resultado inferior ao caso comparado, porém quando positivo significa que a solução é melhor. O **AATS *offline*** foi capaz de encontrar a solução ideal ou uma solução próxima a essa. Considerando a média entre todos os *benchmarks* em cada arquitetura, o **AATS *offline*** ficou com 1,86% mais distante da solução ideal, o que ocorreu na arquitetura AMD. Para as arquiteturas Intel e IBM o **AATS *offline*** ficou com 6,48% e 14,39% distante da solução ideal, respectivamente.

Comparada à forma padrão de execução do OpenMP percebe-se que a solução

Tabela 7 – Diferença da solução e de tempo de aprendizagem do AATS *offline* para Best-DSE

Bench.	***Diferença do AATS offline para Best-DSE***			***Diferença Tempo aprend. para Best-DSE***		
	AMD	***Intel***	***IBM***	***AMD***	***Intel***	***IBM***
FFT	1.26	1.17	1.70	11.4%	4.2%	1.6%
HPCG	1.00	1.09	1.00	10.7%	4.6%	1.8%
JA	1.00	1.30	1.00	7.3%	5.0%	2.1%
LL	1.00	1.00	1.00	11.8%	6.1%	2.1%
BT	1.00	1.01	1.21	8.6%	6.3%	1.5%
CG	1.00	1.03	1.85	9.6%	6.3%	2.3%
EP	1.00	1.00	1.00	11.3%	6.4%	1.8%
FT	1.00	1.00	1.15	9.8%	5.8%	1.7%
LU	1.02	1.06	1.30	11.6%	7.0%	1.7%
MG	1.00	1.20	1.11	17.0%	5.3%	1.5%
SP	1.00	1.00	1.00	10.2%	3.9%	2.2%
UA	1.00	1.04	1.17	9.7%	5.6%	2.1%
ST	1.01	1.07	1.25	9.0%	5.3%	2.5%
Gmean	1.02	1.07	1.19	10.4%	5.4%	1.9%

Fonte: O autor, 2021

Tabela 8 – Diferença das soluções encontradas em relação à Best-DSE e OMP_STD

Bench.	***Diferença do AATS offline para Best-DSE***			***Diferença do AATS offline para OMP_STD***		
	AMD	***Intel***	***IBM***	***AMD***	***Intel***	***IBM***
FFT	-20,77%	-15,12%	-41,47%	169,13%	39,43%	38,11%
HPCG	0,00%	-8,80%	0,00%	28,95%	147,20%	60,32%
JA	0,00%	-23,58%	0,00%	359,21%	134,41%	582,68%
LL	0,00%	0,00%	0,00%	57,29%	363,90%	738,97%
BT	0,00%	-0,12%	-17,43%	192,56%	81,89%	94,05%
CG	0,00%	-3,38%	-46,15%	52,93%	35,04%	-26,88%
EP	0,00%	0,00%	0,00%	1,83%	25,41%	33,95%
FT	0,00%	0,00%	-13,72%	43,87%	32,66%	39,12%
LU	-1,61%	-5,41%	-23,28%	1869,42%	49,60%	182,81%
MG	0,00%	-16,94%	-10,05%	28,41%	107,31%	123,34%
SP	0,00%	0,00%	0,00%	395,00%	623,42%	503,10%
UA	0,00%	-3,95%	-14,60%	362,84%	145,16%	122,11%
ST	-1,77%	-7,00%	-20,37%	899,56%	1219,56%	224,55%
Média	-1,86%	-6,48%	-14,39%	343,15%	231,15%	208,94%

Fonte: O autor, 2021

proposta apresenta grandes percentuais de aprimoramento, apresentando até 343,15% melhor na arquitetura AMD. Para as arquiteturas Intel e IBM atingiu ganho de 231,15% e 208,94%, respectivamente. Observa-se que quanto maior o espaço de exploração maior a diferença da solução encontrada em relação à ***Best-DSE***, além disso a solução encontrada também reduz a distância para a solução ***OMP_STD***. Este fato se deve ao aumento do espaço de exploração e consequentemente maior tempo de aprendizagem.

6.2 Análise da Busca Online

6.2.1 Metodologia da Busca Online

Para avaliação do **AATS *online*** foram utilizados os *benchmarks* apresentados na Seção 4.1.2 e de forma complementar os seguintes *benchmarks*: *Integer Sort* (**IS**) do NAS *Parallel Benchmark*, que realiza a ordenação de números inteiros apresentando acesso aleatório à diferentes áreas da memória; *Hotspot* (**HS**) e *Streamcluster* (**SC**) do Rodinia *Benchmark* (CHE et al., 2009), que realizam, respectivamente, simulação de temperatura de processadores e cálculo de medianas de um conjunto de pontos; e, *Poisson* (**PO**), solução de equações de Poisson para uma área retangular (QUINN, 2004).

A arquitetura utilizada na avaliação é uma máquina com processador AMD Ryzen 9 3900X, que possui 12 *cores* físicos e 24 *threads* de *hardware*, frequência base de 3,8 GHz e *cache* L3 de 64MB total. Os resultados apresentam a comparação entre o **AATS *online*** e a forma padrão de execução do OpenMP (***OMP_STD***).

6.2.2 Resultados Experimentais da Busca Online

De uma maneira geral, os resultados obtidos na utilização do **AATS *online*** mostram que o ajuste do *placement* e da afinidade em tempo de execução impactam negativamente no desempenho e no *aging* comparada à forma padrão de execução.

A Tabela 9 apresenta os dados de desempenho e *aging* para a execução padrão e o **AATS *online***. Os valores de desempenho são relativos ao tempo de execução de cada aplicação, bem como os valores do *aging* que são calculados a partir dos dados coletados. A diferença entre o **AATS *online*** e a forma padrão de execução da aplicação quando utilizado o OpenMP (***OMP_STD***) é apresentada dos pontos de vista do desempenho e do *aging*. Valores positivos indicam que o **AATS *online*** necessitou de mais tempo para a execução (*overhead*) ou que gerou um *aging* maior. Apenas em dois casos o **AATS *online*** apresentou um desempenho melhor que a ***OMP_STD***, sendo eles os *benchmarks* SP e SC na qual a redução foi de 28,18% e 84,65%, respectivamente. Nos demais casos, o tempo necessário para executar os *benchmarks* foi igual ou superior à ***OMP_STD***, chegando até um acréscimo de 7.322,83% no *benchmark* PO. A maior redução no *aging* ocorreu no *benchmark* SC com 18,13% de redução. Ainda, outros quatro *benchmarks* apresentaram redução no *aging* mas com percentuais menores, sendo eles os *benchmarks*

BT, EP, ST e IS. Como reflexo do desempenho, o *benchmark* PO apresentou aumento do *aging* de 41,86%, sendo o pior índice de impacto no *aging*.

Tabela 9 – Diferença das soluções encontradas entre AATS *online* e OMP_STD

Bench.	***Desempenho (s)***		***Aging*** ($\times 10^{-3}$)		***Diferença entre AATS online e OMP_STD***	
	AATS	***OMP_STD***	***AATS***	***OMP_STD***	***Desemp.***	***Aging***
FFT	164,9	164,8	5,24	4,98	0,06%	5,22%
HPCG	116,6	110,9	6,17	5,98	5,14%	3,18%
JA	6,0	6,0	3,55	3,54	0,00%	0,28%
LL	305,2	246,2	6,50	6,32	23,96%	2,85%
BT	92,9	79,5	6,76	6,88	16,86%	-1,74%
CG	38,2	29,0	5,78	4,49	31,72%	5,28%
EP	221,5	221,5	9,42	9,43	0,00%	-0,11%
FT	22,0	18,9	5,52	5,35	16,40%	3,18%
LU	56,4	49,0	5,88	5,69	15,10%	3,34%
MG	20,0	19,0	4,75	4,61	5,26%	3,04%
SP	172,3	239,9	6,62	6,36	-28,18%	4,09%
UA	136,1	102,0	5,76	5,69	33,43%	1,23%
ST	374,0	362,3	6,09	6,11	3,23%	-0,33%
HS	116,0	79,9	7,20	6,70	45,18%	7,46%
SC	76,9	499,1	5,51	6,73	-84,65%	-18,13%
IS	62,0	57,0	4,98	5,04	8,77%	-1,19%
PO	942,7	12,7	6,88	4,85	7322,83%	41,86%

Fonte: O autor, 2021

Com o objetivo de verificar as possíveis causas da ineficácia do **AATS *online***, as aplicações foram monitoradas coletando as seguintes informações:

- ***page-fault:*** contabiliza o número de ocorrências de falta de páginas da memória virtual;
- ***cache-misses:*** contabiliza o número de solicitações dados à cache mas que não estão armazenados nela;
- ***dTLB-load-miss:*** contabiliza as falhas de carregamento de dados da *Translation Lookaside Buffer* (TLB);
- ***iTLB-load-miss:*** contabiliza as falhas de carregamento de instruções da TLB;

Na Tabela 10 está a comparação entre os resultados do **AATS *online*** e o ***OMP_STD***, apresentada na forma da diferença percentual das métricas monitoradas para cada *benchmark*, ou seja, quanto a versão *online* está distante da ***OMP_STD***. Diferenças positivas indicam que a contagem na versão *online* foi maior que no ***OMP_STD***. Por outro lado,

quando os resultados são negativos, indicam contagens menores que o ***OMP_STD***. Os dados foram obtidos utilizando o comando *perf* do Linux.

Tabela 10 – Diferença das métricas de acesso a memória entre AATS *online* e OMP_STD

Bench.	***Métricas***			
	page-fault	***cache-misses***	***dTLB-load-miss***	***iTLB-load-miss***
FFT	0,03%	-52,00%	826,41%	223,00%
HPCG	0,27%	-16,59%	834,33%	1372,69%
JA	0,08%	-57,83%	387,53%	1,35%
LL	-2,12%	-35,45%	693,01%	509,62%
BT	155,90%	-41,82%	522,45%	2916,69%
CG	1,74%	-1,34%	16515,03%	834,13%
EP	-49,58%	15,92%	21693,11%	-72,66%
FT	3,70%	-23,67%	7593,79%	671,06%
LU	5,11%	-23,73%	1185,33%	654,50%
MG	2,25%	-7,78%	486,95%	6345,52%
SP	3338,12%	-61,41%	550,60%	787,56%
UA	267,05%	-31,42%	2212,37%	16000,80%
ST	0,76%	-42,98%	1068,12%	141,37%
HS	1,64%	1,36%	77898,68%	-44,71%
SC	2,98%	4,26%	645,79%	981,47%
IS	0,00%	-8,80%	388,24%	-28,67%
PO	266,94%	112,45%	398317,12%	21404,22%

Fonte: O autor, 2021

A partir dos dados coletados, observa-se maior número de *page-faults* em grande parte das aplicações chegando a um aumento de 3338,12% na aplicação SP. Porém, quando observada na Tabela 9 os valores para a aplicação SP percebe-se que houve um ganho de desempenho do **AATS *online*** comparada à ***OMP_STD***, que sugere que o aumento de *page-faults* não impacta diretamente. A análise do impacto da variação de *cache-misses* também não indica a influência direta no desempenho e no *aging*, pois houveram variações positivas e negativas que não refletiram em variação proporcional do desempenho ou do *aging*.

Os resultados obtidos das falhas de acessos à TLB mostram que houve aumento significativo nas falhas de leitura de dados em todas as aplicações, com o menor aumento em 387,53% para o *benchmark* JA. O maior aumento ocorreu no *benchmark* PO que, juntamente com os dados da variação de desempenho e *aging* da Tabela 9, induz que essa métrica influencia diretamente na eficiência do **AATS *online***. Quando analisada a variação de falhas de leitura de instruções da TLB não é possível afirmar que há influência diretamente proporcional ao desempenho ou *aging*, pois há resultados em que houveram menos falhas de leitura (valores negativos) que não possibilitaram melhor desempenho ou menor *aging* com o **AATS *online***.

Esses resultados caracterizam o impacto do **AATS *online*** cujo ajuste de configurações ocorre em tempo de execução. Por inserir um conjunto de instruções que realiza a busca pela configuração ideal variando o número de *threads*, política de *placement* e estratégia de afinidade, houve um aumento de leituras e escritas na memória. Outro ponto que influencia negativamente é que o algoritmo de busca é executado para cada região paralela de forma independente das demais regiões. Isso implica na maior movimentação de dados pois ao entrar em uma região paralela aplica-se a ela a configuração conhecida até o momento, que poderá ser diferente a da região paralela executada anteriormente. Com isso, variações do número de *threads* farão que a localidade dos dados na memória cache não seja eficiente. O mesmo ocorrerá na aplicação da política de *placement* e da estratégia de afinidade, que irão impactar no compartilhamento ou não dos diferentes níveis de *cache*.

6.3 Busca Offline x Busca Online

Comparando os resultados obtidos na aplicação das duas abordagens destaca-se o seguinte: o **AATS *offline*** não concorre por tempo de processamento pois é executado antes e depois da aplicação, já no o **AATS *online*** há concorrência pelo tempo de processamento já que a cada vez que uma região é executada o algoritmo faz a análise do comportamento, ajustando as configurações. Isso impacta no desempenho, pois a sobrecarga adicionada não é compensada pela otimização do processamento das regiões paralelas, tornando o **AATS *online*** ineficiente.

Os resultados mostram que a sobrecarga do **AATS *online*** são principalmente ocasionados pelo maior acesso à memória. Por trabalhar com uma granularidade fina, pois avalia a região paralela, no ajuste da configuração há movimentação dos dados nos diferentes níveis de memória. Já o **AATS *offline*** não impacta no acesso à memória durante a busca da configuração a ser utilizada, visto que apresenta granularidade grossa e é aplicada antes da execução da aplicação e logo após, processando as informações da eficiência da configuração utilizada nesta execução.

7 CONCLUSÕES

Este trabalho realizou a exploração do espaço de projeto analisando a influência da variação de TLP, políticas de *placement* e estratégias de afinidade de *threads*. Foram avaliadas três políticas e cinco estratégias disponíveis na IPP OpenMP aplicadas a diferentes níveis de TLP. Para isso, foram utilizadas treze aplicações paralelas cujos dados de execução foram coletados em três arquiteturas diferentes.

A partir da análise do espaço de projeto conclui-se que há possibilidade de otimização do tempo de vida dos processadores *multicores*, dependendo da configuração utilizada para o número de *threads*, política de *placement* e estratégia de afinidade. O grau de TLP influencia no desempenho da execução da aplicação, logo no *aging* também. A política de *placement* e estratégia de afinidade exploram a localidade dos dados e compartilhamento da memória *cache* que, conforme as características da aplicação, pode ser benéfico ou não quando as *threads* alocadas compartilham a *cache* de último nível. Neste ponto, a partir dos dados da exploração do espaço, notou-se que a estratégia de afinidade *master* apresentou os piores resultados, pois as *threads* compartilham os mesmos recursos da *thread* principal.

A estratégia de busca apresentada é baseada no algoritmo *hill-climbing* porém com algumas modificações na forma de definição dos passos. A busca da solução ideal é realizada em três fases, iniciando pela variação do grau de TLP, seguido pela variação da política de *placement* e variação da estratégia de afinidade, buscando maximizar a métrica de avaliação. O algoritmo de busca foi utilizado em duas abordagens diferentes: uma *offline* cujo processo de avaliação ocorre externamente à aplicação; e, um *online* na qual avalia as regiões paralelas durante a execução da aplicação.

O **AATS *offline*** apresentou bons resultados sendo possível reduzir o *aging* usando o aprendizado após cada execução. Considerando a média geométrica entre todos os *benchmarks* em cada arquitetura o **AATS *offline*** ficou com até 1,19 vezes mais distante da solução ideal para uma arquitetura. Observa-se também que com o **AATS *offline*** para chegar a solução foi necessário avaliar, no máximo, somente 10,4% das possíveis configurações de uma arquitetura, considerando a média geométrica dos *benchmarks*.

Quando avaliado o **AATS *online***, os resultados indicam que o algoritmo não foi eficiente na busca de uma configuração ideal, causando redução do desempenho e aumentando o *aging* na maioria dos *benchmarks*. Com base nos dados apresentados sobre as métricas de acesso à memória concluiu-se que houve grande aumento das falhas de leitura de dados da TLB, impactando negativamente no desempenho das aplicações. Também mostrou-se que as outras métricas, que embora tenham sofrido variação devido ao **AATS *online***, não apresentaram relação direta com o desempenho e *aging*.

7.1 Publicações

Durante a pesquisa, alguns trabalhos foram submetidos e aceitos para publicação. São eles:

- ***The Impact of Turbo Frequency on the Energy, Performance, and Aging of Parallel Applications*** - VLSI-SOC (Qualis: B2) (MARQUES et al., 2019) avalia o impacto da alteração das configurações de quantidade de *threads*, SMT, reguladores de escalonamento dinâmico de voltagem e frequência e tecnologias de *boosting* no consumo de energia, desempenho e envelhecimento de aplicativos paralelos em um processador com *Turbo Core*;

- ***Transparent Aging-Aware Thread Throttling*** - SBAC-PAD (Qualis A4) (MEDEIROS et al., 2019) propõe uma uma abordagem automática e transparente para reduzir o envelhecimento do processador, ajustando automaticamente o número de *threads* para aplicativos OpenMP em tempo de execução. Apresenta a ferramenta Geras, que é totalmente transparente para o usuário final onde mesmo binários já compilados podem ser otimizados;

- **Impacto do Mapeamento de threads na Degradação do Processador** - ERAD-RS (MEDEIROS; LORENZON, 2020) apresenta, resumidamente, os primeiros resultados obtidos nas execuções dos *benchmarks*, relatando as primeiras impressões do comportamento do *aging* conforme as configurações utilizadas de TLP, *placement* e afinidade de *threads*.

- ***Aging-aware Parallel Execution*** - *IEEE Embedded Systems Letters (ESL)* (Qualis A2) (MEDEIROS et al., 2020), na qual avalia qual IPPs e quantas *threads* utilizar na execução de aplicações paralelas, minimizando o *aging*;

- ***An Application-Driven Approach to Mitigate Aging by Tuning the TLP and Allocation Strategies*** - *International Conferences on High Performance Computing and Communications (HPCC)* (Qualis A3) (MEDEIROS et al., 2020), ampliando o número de arquiteturas utilizadas, empregando o modelo mais completo de cálculo da NBTI e propondo uma estratégia para definição do grau de TLP e alocação de *threads* minimizando o *aging*.

- ***Combining Thread Throttling and Mapping to Optimize the EDP of Parallel Applications*** - *Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)* (Qualis A3) (BERNED et al., 2021), otimiza o EDP (*Energy Delay Product*) utilizando a estratégia para definição do grau de TLP e alocação de *threads*.

- ***Synergically Rebalancing Parallel Execution via DCT and Turbo Boosting*** - *Design Automation Conference (DAC)* (Qualis A1), além de utilizar o ajuste de TLP também ajusta a frequência de operação.

Ainda, há artigo em revisão para o periódico:

- ***Mitigating the Processor Aging through Dynamic Concurrency Throttling*** - *Journal of Parallel and Distributed Computing (JPDC)* (Qualis A2), estendendo o trabalho apresentado no SBAC-PAD 2019, utilizando a NBTI para cálculo do *aging*;

7.2 Trabalhos Futuros

Para trabalhos futuros, destacam-se as seguintes propostas

- otimizar a utilização do **AATS *offline***, buscando reduzir o tempo de aprendizagem pelo emprego de estratégias como redes neurais para inferir a configuração que possibilite o menor *aging*;
- implementar o **AATS *offline*** internamente na biblioteca do OpenMP deixando transparente para o usuário, não sendo necessário executar uma aplicação externa;
- implementar no **AATS *online*** o armazenamento das configurações por região paralela, reduzindo o custo da aprendizagem nas execuções futuras, além de ser possível compensar o tempo de aprendizagem nas execuções seguintes;
- elaborar políticas de alocação de *threads* que possibilitem a exploração da localidade dos dados nas memórias *cache* mantendo o desempenho e reduzindo o *aging*, porém sem aumentar o custo de processamento ou de leitura de dados da TLB.
- unificar as duas abordagens um uma única biblioteca, podendo o usuário definir qual deseja aplicar e ainda haver compartilhamento dos dados entre as abordagens, como por exemplo usar a configuração obtida no **AATS *offline*** como configuração inicial das regiões paralelas quando usando o **AATS *online***.

REFERÊNCIAS

ALESSANDRINI, V. Chapter 10 - openmp. In: ALESSANDRINI, V. (Ed.). **Shared Memory Application Programming**. Boston: Morgan Kaufmann, 2016. p. 225 – 305. ISBN 978-0-12-803761-4. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/B9780128037614000101⟩.

ALESSI, F. et al. Application-level energy awareness for openmp. In: TERBOVEN, C. et al. (Ed.). **OpenMP: Heterogenous Execution and Data Movements**. Cham: Springer International Publishing, 2015. p. 219–232. ISBN 978-3-319-24595-9.

Amrouch, H. et al. Towards interdependencies of aging mechanisms. In: **2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)**. [S.l.: s.n.], 2014. p. 478–485.

BAILEY, D. H. et al. The NAS parallel benchmarks – summary and preliminary results. In: **ACM/IEEE CS**. NY, USA: ACM, 1991. p. 158–165. ISBN 0-89791-459-7.

BASOGLU, M.; ORSHANSKY, M.; EREZ, M. Nbti-aware dvfs: A new approach to saving energy and increasing processor lifetime. In: **2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)**. [S.l.: s.n.], 2010. p. 253–258. ISSN null.

BERNED, G. P. et al. Combining thread throttling and mapping to optimize the edp of parallel applications. In: **2021 29th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)**. [S.l.: s.n.], 2021. p. 177–180.

BHARDWAJ, S. et al. Predictive modeling of the nbti effect for reliable design. In: **IEEE Custom Integrated Circuits Conference 2006**. [S.l.: s.n.], 2006. p. 189–192.

BLAKE, G. et al. Evolution of thread-level parallelism in desktop applications. **SIGARCH Comput. Archit. News**, ACM, NY, USA, v. 38, n. 3, p. 302–313, 2010. ISSN 0163-5964.

BLAT, C.; NICOLLIAN, E.; POINDEXTER, E. Mechanism of negative-bias-temperature instability. **Journal of Applied Physics**, American Institute of Physics, v. 69, n. 3, p. 1712–1720, 1991.

CHANTEM, T.; HU, X. S.; DICK, R. P. Temperature-aware scheduling and assignment for hard real-time applications on mpsocs. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 19, n. 10, p. 1884–1897, 2011.

CHANTEM, T. et al. Enhancing multicore reliability through wear compensation in online assignment and scheduling. In: **2013 Design, Automation Test in Europe Conference Exhibition (DATE)**. [S.l.: s.n.], 2013. p. 1373–1378. ISSN 1530-1591.

CHE, S. et al. Rodinia: A benchmark suite for heterogeneous computing. In: **2009 IEEE International Symposium on Workload Characterization (IISWC)**. [S.l.: s.n.], 2009. p. 44–54.

CHIEN, T.-H.; CHANG, R.-G. A thermal-aware scheduling for multicore architectures. **Journal of Systems Architecture**, v. 62, p. 54 – 62, 2016. ISSN 1383-7621. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S1383762115001538⟩.

COCHRAN, R. et al. Pack cap: Adaptive dvfs and thread packing under power caps. In: **2011 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)**. [S.l.: s.n.], 2011. p. 175–185.

CORBETTA, S.; FORNACIARI, W. Nbti mitigation in microprocessor designs. In: **Proceedings of the Great Lakes Symposium on VLSI**. New York, NY, USA: Association for Computing Machinery, 2012. (GLSVLSI '12), p. 33–38. ISBN 9781450312448. Disponível em: ⟨https://doi.org/10.1145/2206781.2206791⟩.

CORMEN, T. H. et al. **Introduction to Algorithms, Third Edition**. 3rd. ed. [S.l.]: The MIT Press, 2009. ISBN 0262033844.

CRUZ, E. H. M. D. et al. Using memory access traces to map threads and data on hierarchical multi-core platforms. In: **2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum**. [S.l.: s.n.], 2011. p. 551–558.

CURTIS-MAURY, M. et al. Prediction models for multi-dimensional power-performance optimization on many cores. In: **2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)**. [S.l.: s.n.], 2008. p. 250–259.

DANELUTTO, M. et al. Evaluating concurrency throttling and thread packing on smt multicores. In: **2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP)**. [S.l.: s.n.], 2017. p. 219–223.

DASHTI, M. et al. Traffic management: A holistic approach to memory placement on numa systems. In: **Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems**. New York, NY, USA: Association for Computing Machinery, 2013. (ASPLOS '13), p. 381–394. ISBN 9781450318709. Disponível em: ⟨https://doi.org/10.1145/2451116.2451157⟩.

DIAZ, J.; MUÑOZ-CARO, C.; NIÑO, A. A survey of parallel programming models and tools in the multi and many-core era. **IEEE Transactions on Parallel and Distributed Systems**, v. 23, n. 8, p. 1369–1386, 2012.

DIENER, M. et al. Kernel-based thread and data mapping for improved memory affinity. **IEEE Transactions on Parallel and Distributed Systems**, v. 27, n. 9, p. 2653–2666, 2016.

DIENER, M. et al. kmaf: Automatic kernel-level management of thread and data affinity. In: **2014 23rd International Conference on Parallel Architecture and Compilation Techniques (PACT)**. [S.l.: s.n.], 2014. p. 277–288.

DONGARRA, J.; HEROUX, M. A.; LUSZCZEK, P. Hpcg benchmark: A new metric for ranking high performance computing systems. **Knoxville, Tennessee**, 2015.

EGGERS, S. J. et al. Simultaneous multithreading: a platform for next-generation processors. **IEEE Micro**, v. 17, n. 5, p. 12–19, 1997.

FENG, H. et al. **Unstructured Adaptive (UA) NAS Parallel Benchmark, Version 1.0**. [S.l.], 2004. 1-17 p.

FLAUTNER, K. et al. Thread-level parallelism and interactive performance of desktop applications. In: **Proceedings of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems**. New York, NY, USA: Association for Computing Machinery, 2000. (ASPLOS IX), p. 129–138. ISBN 1581133170. Disponível em: ⟨https://doi.org/10.1145/378993.379233⟩.

GÖS, W. **Hole trapping and the negative bias temperature instability**. Dissertação (Mestrado) — Technischen Universität Wien Fakultät für Elektrotechnik und Informationstechnik, 12 2011.

HANUMAIAH, V. et al. Throughput optimal task allocation under thermal constraints for multi-core processors. In: **2009 46th ACM/IEEE Design Automation Conference**. [S.l.: s.n.], 2009. p. 776–781. ISSN 0738-100X.

HENKEL, J. et al. Thermal management for dependable on-chip systems. In: **2013 18th Asia and South Pacific Design Automation Conference (ASP-DAC)**. [S.l.: s.n.], 2013. p. 113–118.

HUANG, H. et al. Throughput maximization for periodic real-time systems under the maximal temperature constraint. In: **2011 48th ACM/EDAC/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2011. p. 363–368.

HYDRODYNAMICS Challenge Problem, Lawrence Livermore National Laboratory. [S.l.], 2011. 1-17 p.

JIN, H.; FRUMKIN, M.; YAN, J. **The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance**. [S.l.], 1999. 1-26 p.

KHDR, H.; AMROUCH, H.; HENKEL, J. Aging-constrained performance optimization for multi cores. In: **2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2018. p. 1–6. ISSN null.

KHDR, H. et al. mdtm: Multi-objective dynamic thermal management for on-chip systems. In: **2014 Design, Automation Test in Europe Conference Exhibition (DATE)**. [S.l.: s.n.], 2014. p. 1–6.

KIM, H. et al. Use it or lose it: Wear-out and lifetime in future chip multiprocessors. In: **2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)**. [S.l.: s.n.], 2013. p. 136–147. ISSN null.

KIRK, D. B.; HWU, W. mei W. **Programming Massively Parallel Processors (Third Edition)**. Third edition. Morgan Kaufmann, 2017. ISBN 978-0-12-811986-0. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/B9780128119860000017⟩.

KUMAR, R. et al. Single-isa heterogeneous multi-core architectures: the potential for processor power reduction. In: **Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.** [S.l.: s.n.], 2003. p. 81–92.

LEE, H.; SHAFIQUE, M.; FARUQUE, M. A. A. Aging-aware workload management on embedded gpu under process variation. **IEEE Transactions on Computers**, v. 67, n. 7, p. 920–933, 2018.

LEE, J. et al. Thread tailor: Dynamically weaving threads together for efficient, adaptive parallel applications. In: **Proceedings of the 37th Annual International Symposium on Computer Architecture**. New York, NY, USA: Association for Computing Machinery, 2010. (ISCA '10), p. 270–279. ISBN 9781450300537. Disponível em: ⟨https://doi.org/10.1145/1815961.1815996⟩.

LERNER, S.; TASKIN, B. Workload-aware asic flow for lifetime improvement of multi-core iot processors. In: **2017 18th International Symposium on Quality Electronic Design (ISQED)**. [S.l.: s.n.], 2017. p. 379–384. ISSN 1948-3287.

LI, D. et al. Hybrid mpi/openmp power-aware computing. In: **2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)**. [S.l.: s.n.], 2010. p. 1–12.

LI, J.; MARTINEZ, J. F. Dynamic power-performance adaptation of parallel computation on chip multiprocessors. In: **The Twelfth International Symposium on High-Performance Computer Architecture, 2006.** [S.l.: s.n.], 2006. p. 77–87.

LIU, J. C. et al. A reliability enhanced 5nm cmos technology featuring 5th generation finfet with fully-developed euv and high mobility channel for mobile soc and high performance computing application. In: **2020 IEEE International Electron Devices Meeting (IEDM)**. [S.l.: s.n.], 2020. p. 9.2.1–9.2.4.

LORENZON, A. F.; FILHO, A. C. S. beck. **Parallel Computing Hits the Power Wall: Principles, Challenges, and a Survey of Solutions**. [S.l.]: Springer, 2019.

MARQUES, S. M. V. N. et al. The impact of turbo frequency on the energy, performance, and aging of parallel applications. In: **2019 IFIP/IEEE 27th International Conference on Very Large Scale Integration (VLSI-SoC)**. [S.l.: s.n.], 2019. p. 149–154.

MCCALPIN, J. D. Memory bandwidth and machine balance in current high performance computers. **IEEE Computer Society Technical Committee on Computer Architecture Newsletter**, p. 19–25, 1995.

MEDEIROS, T. S. et al. Aging-aware parallel execution. **IEEE Embedded Systems Letters**, p. 1–1, 2020.

MEDEIROS, T. S.; LORENZON, A. F. Impacto do mapeamento de threads na degradação do processador. In: **2020: Anais da XX Escola Regional de Alto Desempenho da Região Sul**. [S.l.: s.n.], 2020. p. 173–174.

MEDEIROS, T. S. et al. Transparent aging-aware thread throttling. In: **2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)**. [S.l.: s.n.], 2019. p. 1–8.

MEDEIROS, T. S. et al. An application-driven approach to mitigate aging by tuning the tlp and allocation strategies. In: **2020 IEEE 22th International Conference on High Performance Computing and Communications (HPCC)**. [S.l.: s.n.], 2020. p. 1–9.

MERCATI, P. et al. Warm: Workload-aware reliability management in linux/android. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 36, n. 9, p. 1557–1570, 2017.

MISHRA, N. et al. A probabilistic graphical model-based approach for minimizing energy under performance constraints. In: **Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems**. New York, NY, USA: Association for Computing Machinery, 2015. (ASPLOS '15), p. 267–281. ISBN 9781450328357. Disponível em: ⟨https://doi.org/10.1145/2694344.2694373⟩.

MULAS, F. et al. Thermal balancing policy for multiprocessor stream computing platforms. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 28, n. 12, p. 1870–1882, 2009.

MüCK, T. R. et al. Exploiting heterogeneity for aging-aware load balancing in mobile platforms. **IEEE Transactions on Multi-Scale Computing Systems**, v. 3, n. 1, p. 25–35, Jan 2017. ISSN 2372-207X.

NAMAZI, A. et al. Lrtm: Life-time and reliability-aware task mapping approach for heterogeneous multi-core systems. In: **2018 11th International Workshop on Network on Chip Architectures (NoCArc)**. [S.l.: s.n.], 2018. p. 1–6. ISSN null.

OBORIL, F.; TAHOORI, M. B. Extratime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level. In: **IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)**. [S.l.: s.n.], 2012. p. 1–12.

OPENMP API Specification. 2018. ⟨https://www.openmp.org/spec-html/5.0/openmp.html⟩. Acessado: 2020-03-10.

PAGANI, S. et al. Matex: Efficient transient and peak temperature computation for compact thermal models. In: **DATE**. [S.l.: s.n.], 2015. p. 1515–1520.

PETERSEN, W.; ARBENZ, P. **Introduction to Parallel Computing: A practical guide with examples in C**. [S.l.]: OUP Oxford, 2004. (Oxford Texts in Applied and Engineering Mathematics). ISBN 9780191513619.

QUINN, M. **Parallel Programming in C with MPI and OpenMP**. [S.l.]: McGraw-Hill Higher Education, 2004. ISBN 9780072822564.

RATHORE, V. et al. Himap: A hierarchical mapping approach for enhancing lifetime reliability of dark silicon manycore systems. In: **2018 Design, Automation Test in Europe Conference Exhibition (DATE)**. [S.l.: s.n.], 2018. p. 991–996.

RATHORE, V. et al. Life guard: A reinforcement learning-based task mapping strategy for performance-centric aging management. In: **2019 56th ACM/IEEE Design Automation Conference (DAC)**. [S.l.: s.n.], 2019. p. 1–6. ISSN 0738-100X.

RAUBER, T.; RÜNGER, G. **Parallel Programming - for Multicore and Cluster Systems.** [S.l.]: Springer, 2010. ISBN 978-3-642-04817-3.

REGHENZANI, F. et al. A constrained extremum-seeking control for cpu thermal management. In: **Proceedings of the 15th ACM International Conference on Computing Frontiers**. New York, NY, USA: Association for Computing Machinery, 2018. (CF '18), p. 320–325. ISBN 9781450357616. Disponível em: ⟨https://doi.org/10.1145/3203217.3204464⟩.

ROZO, L. et al. Reliability-aware runtime adaption through a statically generated task schedule. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 26, n. 1, p. 11–22, Jan 2018. ISSN 1557-9999.

SAWALHA, L.; TULL, M. P.; BARNES, R. D. Thread scheduling for heterogeneous multicore processors using phase identification. **SIGMETRICS Perform. Eval. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 39, n. 3, p. 125–127, dez. 2011. ISSN 0163-5999. Disponível em: ⟨https://doi.org/10.1145/2160803.2160879⟩.

SCHRODER, D. K.; BABCOCK, J. A. Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing. **Journal of applied Physics**, American Institute of Physics, v. 94, n. 1, p. 1–18, 2003.

SENSI, D. D.; TORQUATI, M.; DANELUTTO, M. A reconfiguration algorithm for power-aware parallel applications. **ACM Transactions on Architecture and Code Optimization**, v. 13, p. 1–25, 12 2016.

SEO, S.; JO, G.; LEE, J. Performance characterization of the NAS parallel benchmarks in OpenCL. In: **IEEE ISWC**. [S.l.: s.n.], 2011. p. 137–148.

SHAFIK, R. et al. Thermal-aware adaptive energy minimization of openmp parallel applications. In: **DATE2015: Workshop on Designing with Uncertainty - Opportunities & Challenges**. [S.l.: s.n.], 2015. p. 1–3.

SRIDHARAN, S.; GUPTA, G.; SOHI, G. S. Adaptive, efficient, parallel execution of parallel programs. In: **ACM Programming Language Design and Implementation (PLDI)**. USA: ACM, 2014. p. 169–180.

STALLINGS, W. **Arquitetura e Organização de Computadores**. São Paulo: Pearson Education do Brasil, 2017.

STATHIS, J. H.; ZAFAR, S. The negative bias temperature instability in mos devices: A review. **Microelectronics Reliability**, Elsevier, v. 46, n. 2-4, p. 270–286, 2006.

SULEMAN, M. A.; QURESHI, M. K.; PATT, Y. N. Feedback-driven threading: Power-efficient and high-performance execution of multi-threaded workloads on cmps. **SIGOPS Oper. Syst. Rev.**, Association for Computing Machinery, New York, NY, USA, v. 42, n. 2, p. 277–286, mar. 2008. ISSN 0163-5980. Disponível em: ⟨https://doi.org/10.1145/1353535.1346317⟩.

TU, J. et al. Particle swarm optimization based task scheduling for multi-core systems under aging effect. In: **2017 International Conference on Progress in Informatics and Computing (PIC)**. [S.l.: s.n.], 2017. p. 271–276. ISSN null.

USING KMP_AFFINITY to create OpenMP thread mapping to OS proc IDs. 2012. ⟨https://software.intel.com/content/www/us/en/develop/articles/using-kmp-affinity-to-create-openmp-thread-mapping-to-os-proc-ids.html⟩. Acessado: 2020-06-13.

VIEIRA, A. N. C. et al. The impact of parallel programming interfaces on the aging of a multicore embedded processor. In: **2019 IEEE International Symposium on Circuits and Systems (ISCAS)**. [S.l.: s.n.], 2019. p. 1–5. ISSN 2158-1525.

WHITE, M.; BERNSTEIN, J. B. **Microelectronics reliability : physics-of-failure based modeling and lifetime evaluation**. Pasadena, California, 2008.

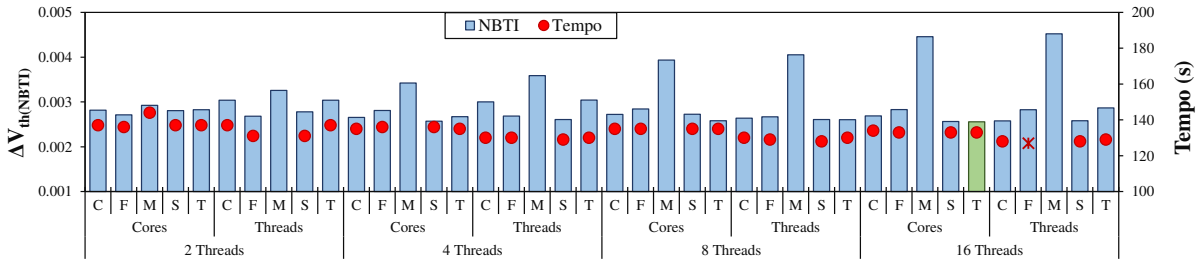
YEO, Y. I.; LIU, C. C.; KIM, E. J. Predictive dynamic thermal management for multicore systems. In: **2008 45th ACM/IEEE Design Automation Conference**. [S.l.: s.n.], 2008. p. 734–739. ISSN 0738-100X.

ZONI, D.; FORNACIARI, W. Nbti-aware design of noc buffers. In: **Proceedings of the 2013 Interconnection Network Architecture: On-Chip, Multi-Chip**. New York, NY, USA: Association for Computing Machinery, 2013. (IMA-OCMC '13), p. 25–28. ISBN 9781450317849. Disponível em: ⟨https://doi.org/10.1145/2482759.2482766⟩.

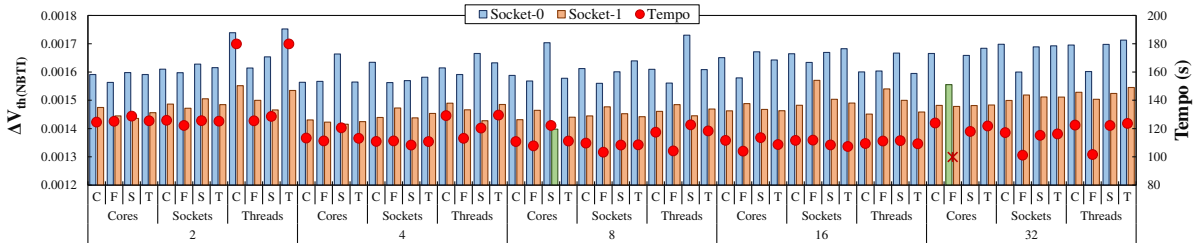
APÊNDICE A – RESULTADOS DA EXPLORAÇÃO DE ESPAÇO E PROJETO

Nesta Seção, apresentamos os resultados para cada aplicação para cada processador com diferentes números de *threads* e políticas de *placement* e afinidade de *threads*. Para auxiliar na identificação da melhor configuração de tempo de execução e NBTI, as mesmas estão destacadas no gráfico: barra verde apresenta a melhor NBTI e o marcador X destaca a configuração com o melhor tempo de execução.

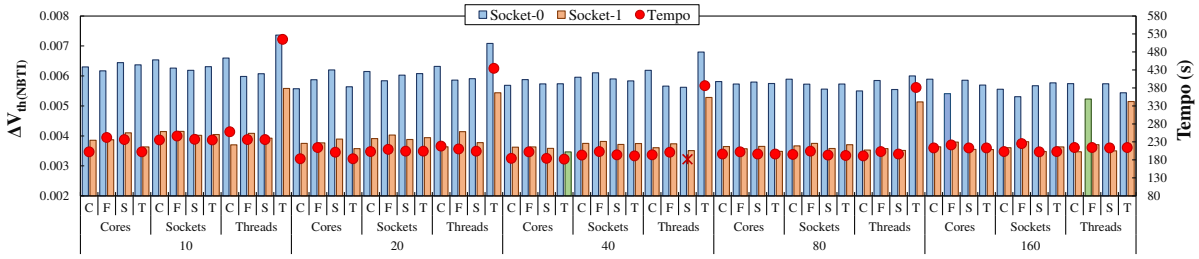
Figura 20 – Alta taxa de comunicação: FFT



(a) AMD Ryzen 7 16-core



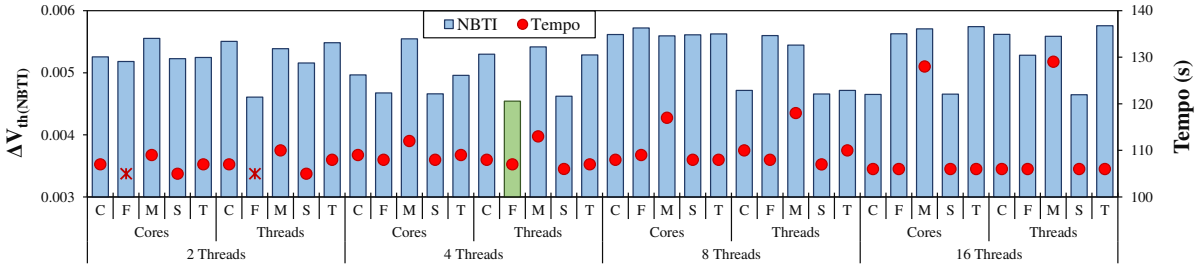
(b) Intel Xeon 32-core



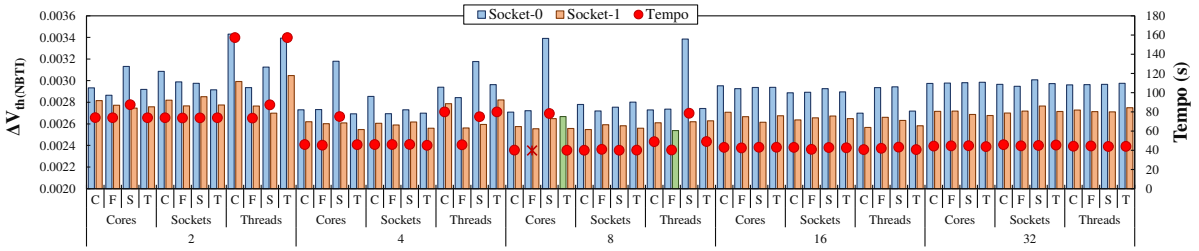
(c) IBM Power9

Fonte: O autor, 2021

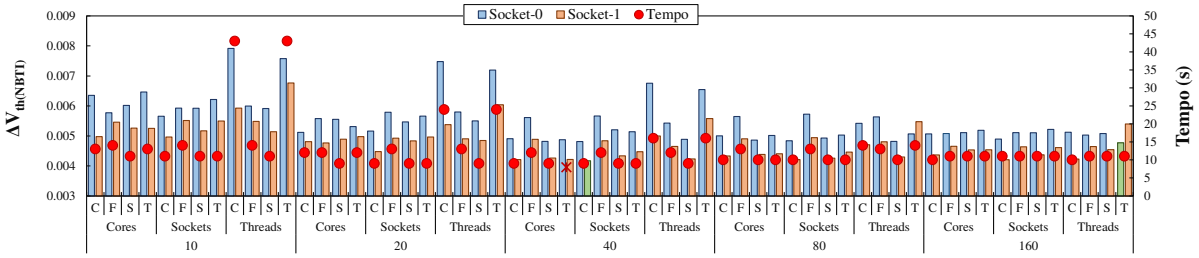
Figura 21 – Alta taxa de comunicação: Jacobi



(a) AMD Ryzen 7 16-core



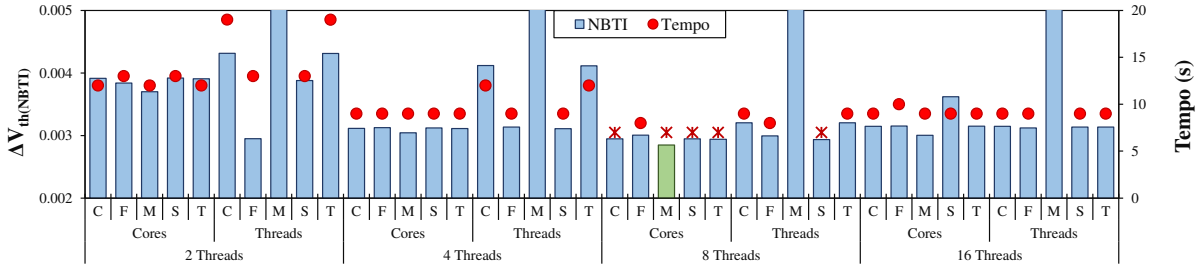
(b) Intel Xeon 32-core



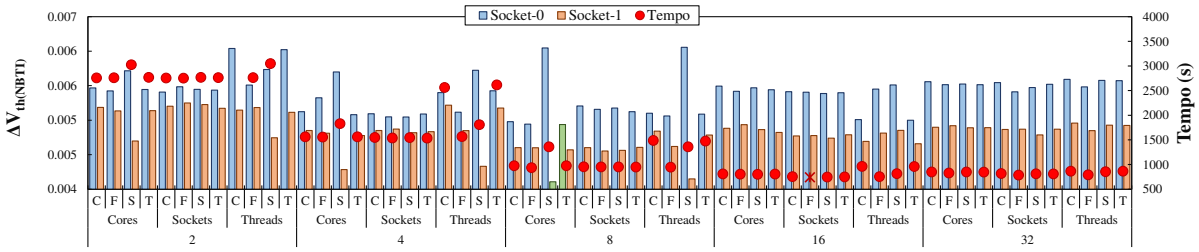
(c) IBM Power9

Fonte: O autor, 2021

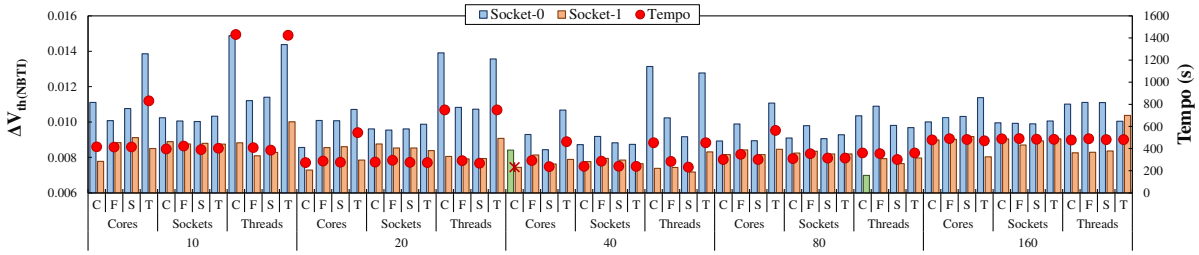
Figura 22 – Alta taxa de comunicação: LULESH2.0



(a) AMD Ryzen 7 16-core



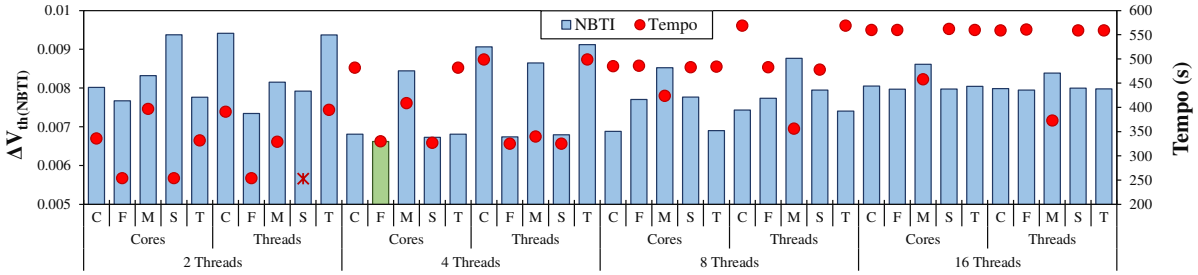
(b) Intel Xeon 32-core



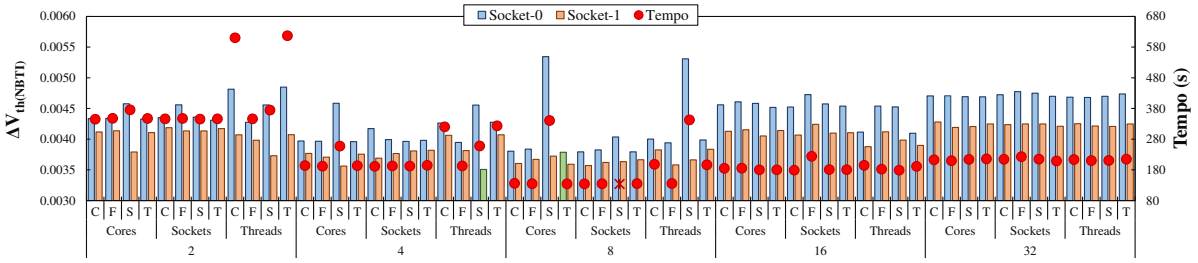
(c) IBM Power9

Fonte: O autor, 2021

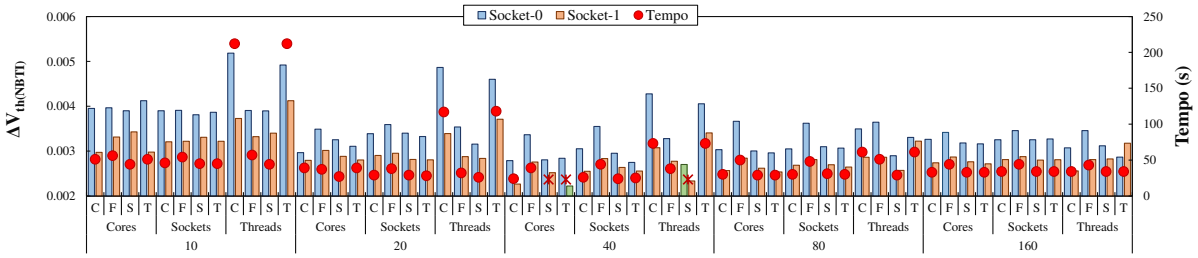
Figura 23 – Alta taxa de comunicação: sp.C.x



(a) AMD Ryzen 7 16-core



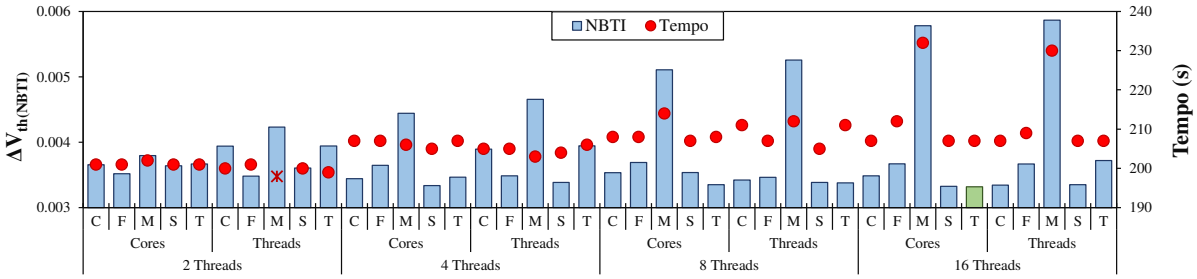
(b) Intel Xeon 32-core



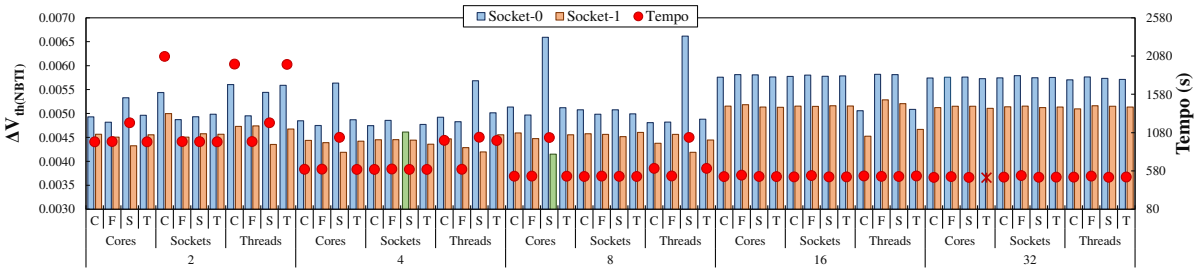
(c) IBM Power9

Fonte: O autor, 2021

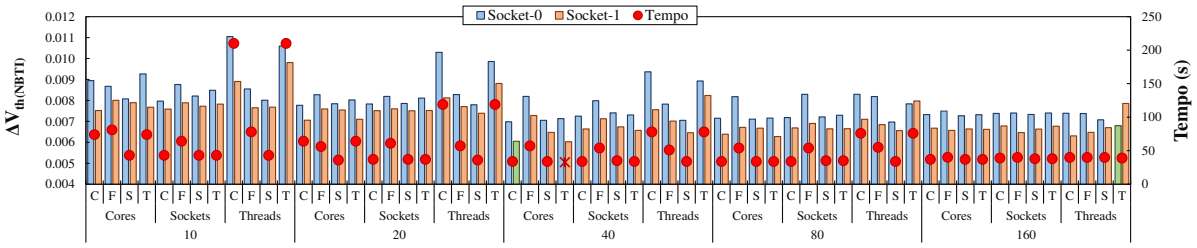
Figura 24 – Alta taxa de comunicação: STREAM



(a) AMD Ryzen 7 16-core



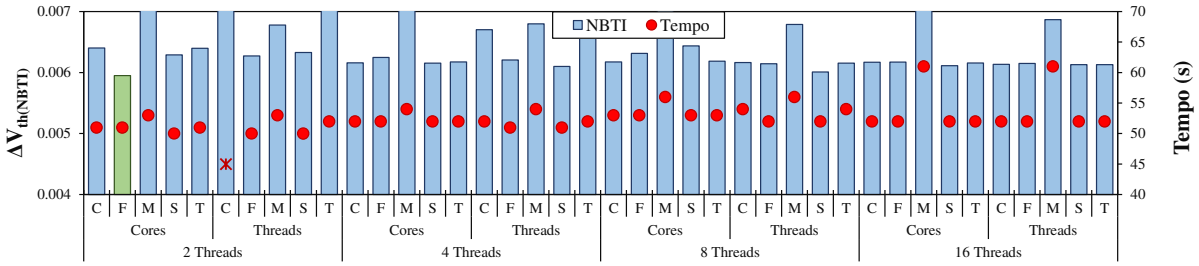
(b) Intel Xeon 32-core



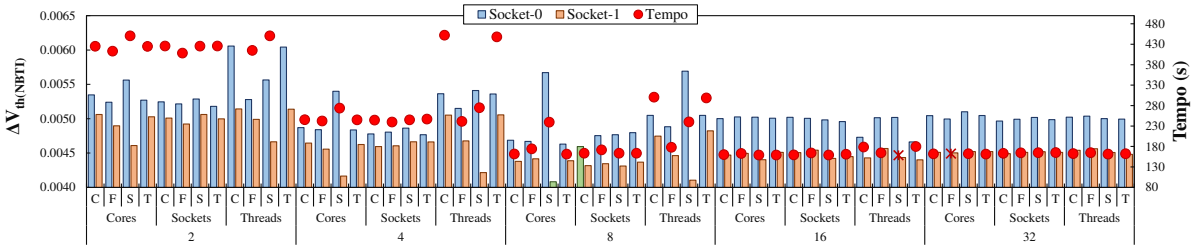
(c) IBM Power9

Fonte: O autor, 2021

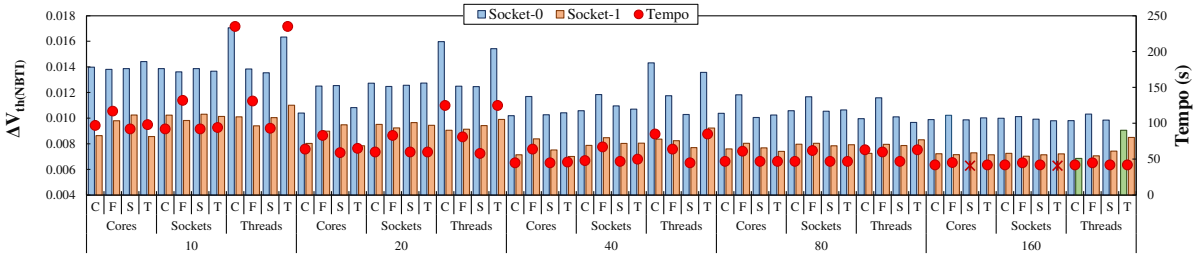
Figura 25 – Baixa taxa de comunicação: HPCG



(a) AMD Ryzen 7 16-core



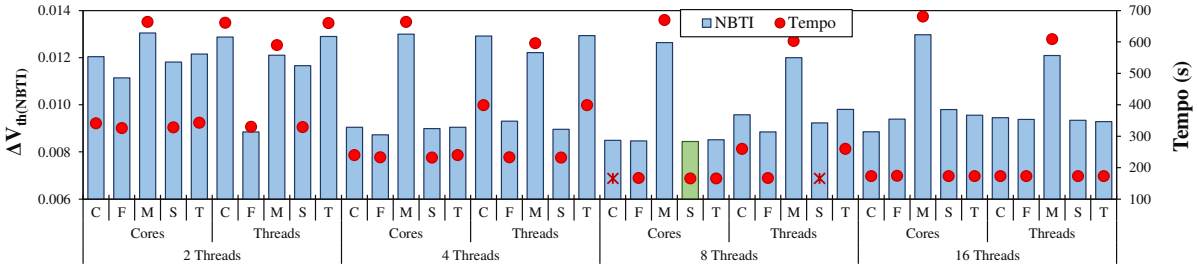
(b) Intel Xeon 32-core



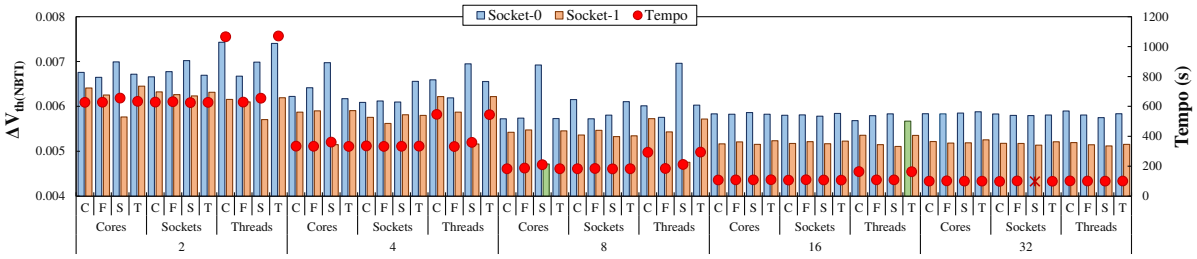
(c) IBM Power9

Fonte: O autor, 2021

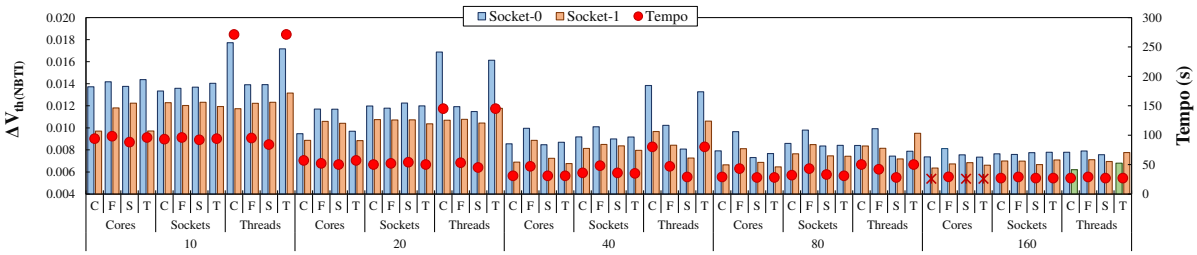
Figura 26 – Baixa taxa de comunicação: bt.C.x



(a) AMD Ryzen 7 16-core



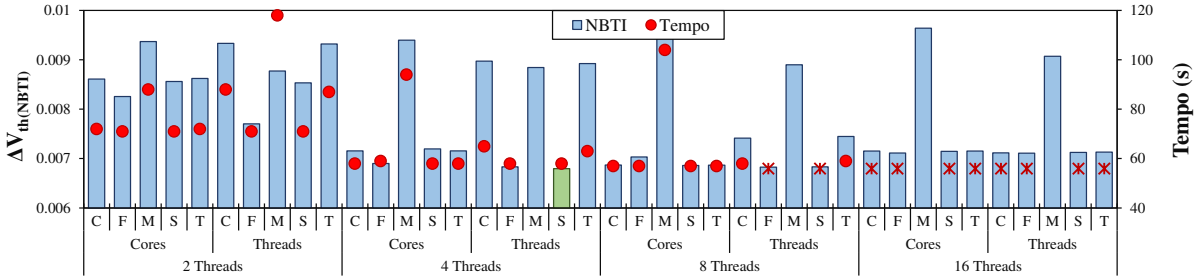
(b) Intel Xeon 32-core



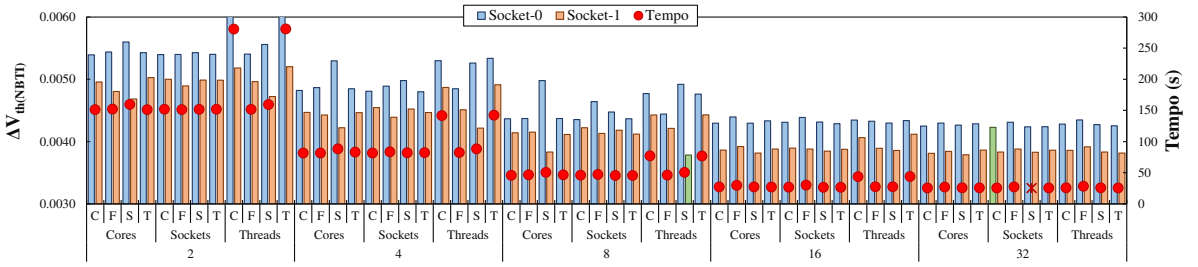
(c) IBM Power9

Fonte: O autor, 2021

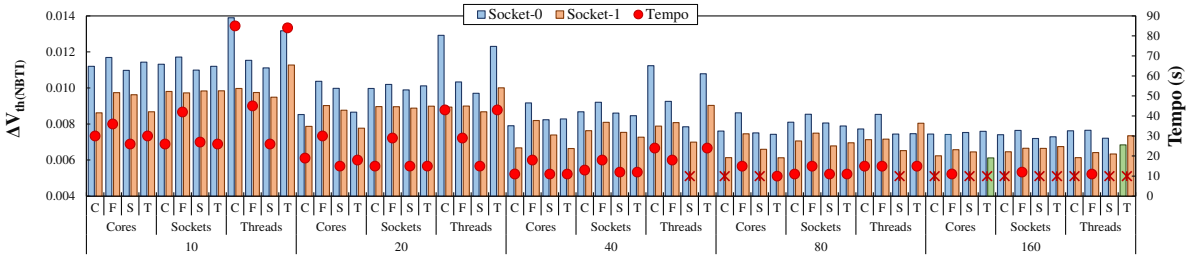
Figura 27 – Baixa taxa de comunicação: cg.C.x



(a) AMD Ryzen 7 16-core



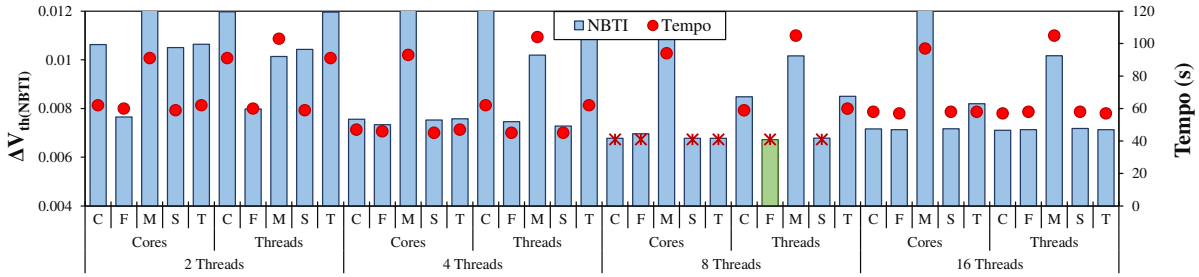
(b) Intel Xeon 32-core



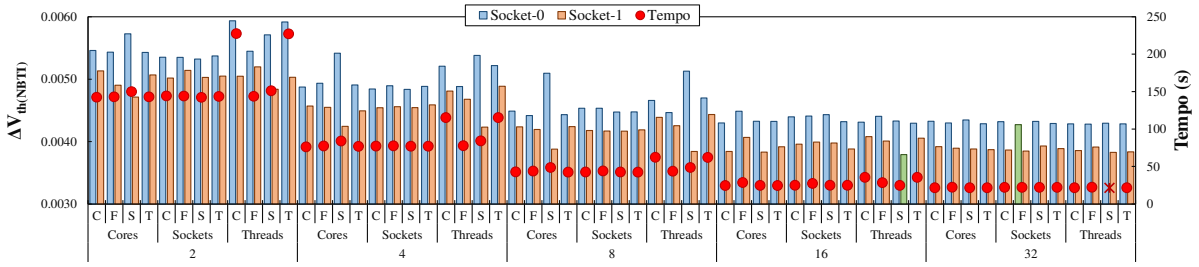
(c) IBM Power9

Fonte: O autor, 2021

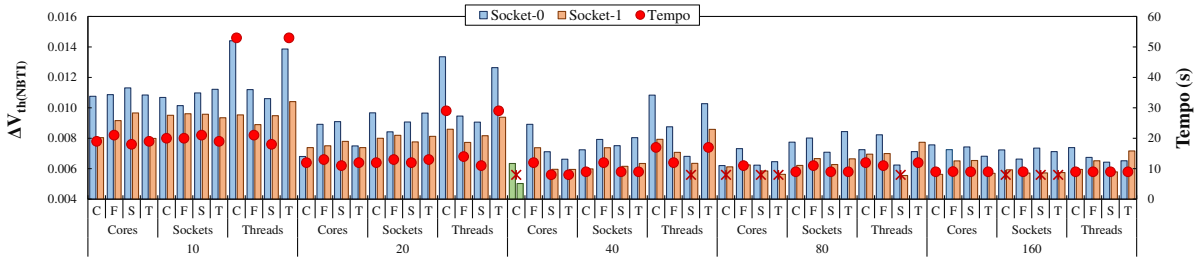
Figura 28 – Baixa taxa de comunicação: ft.C.x



(a) AMD Ryzen 7 16-core



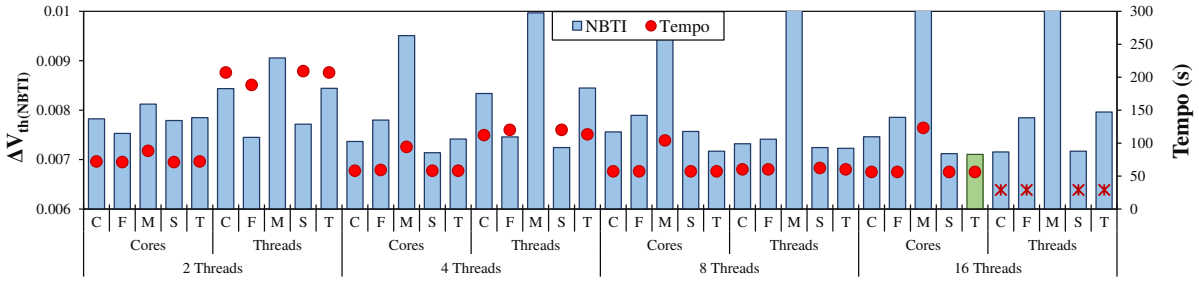
(b) Intel Xeon 32-core



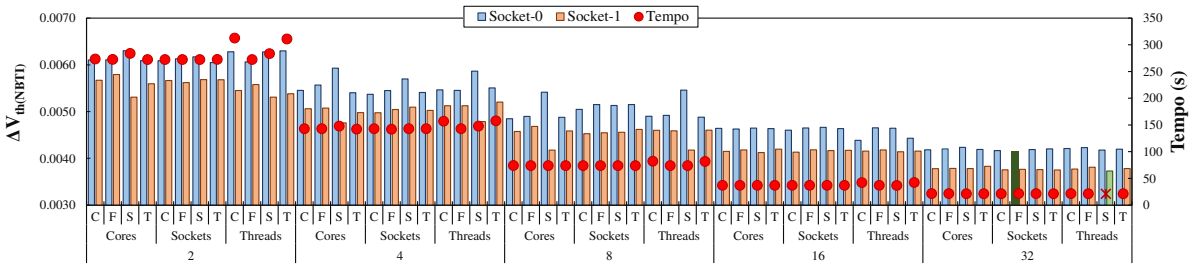
(c) IBM Power9

Fonte: O autor, 2021

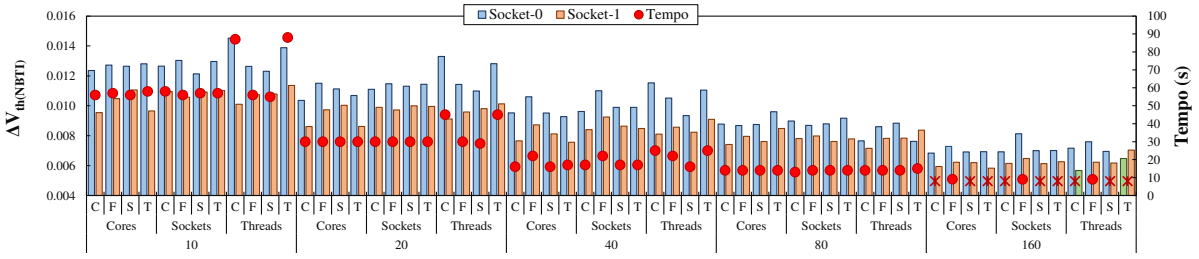
Figura 29 – Baixa taxa de comunicação: ep.C.x



(a) AMD Ryzen 7 16-core



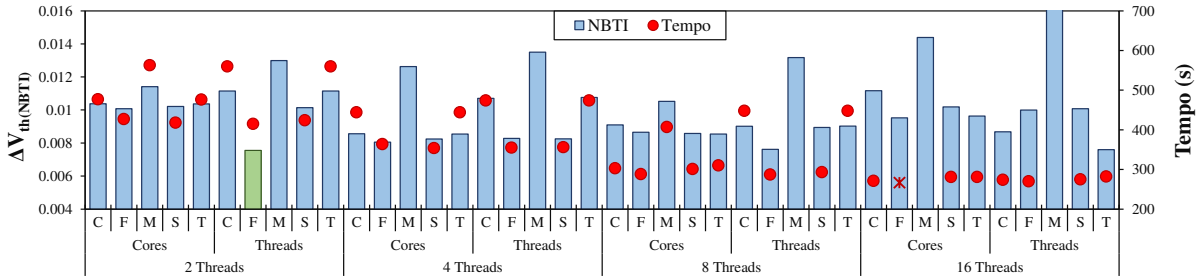
(b) Intel Xeon 32-core



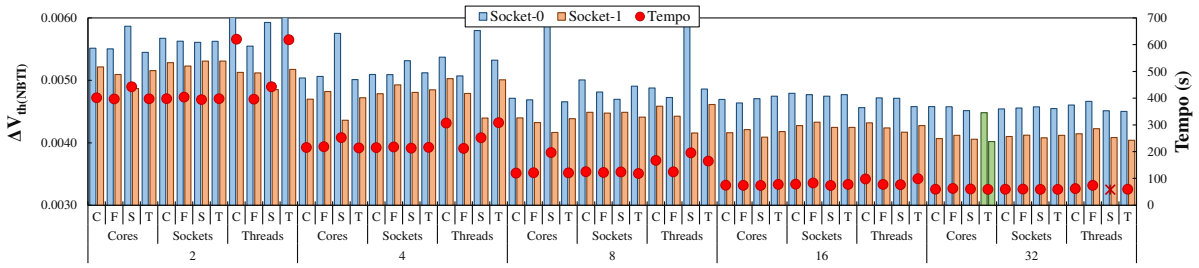
(c) IBM Power9

Fonte: O autor, 2021

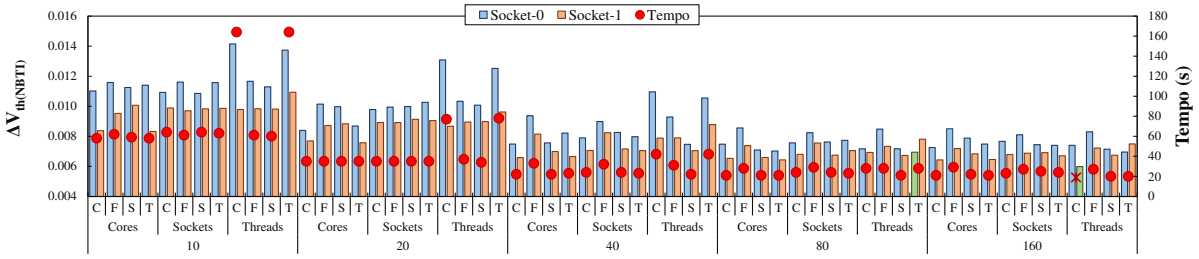
Figura 30 – Baixa taxa de comunicação: lu.C.x



(a) AMD Ryzen 7 16-core



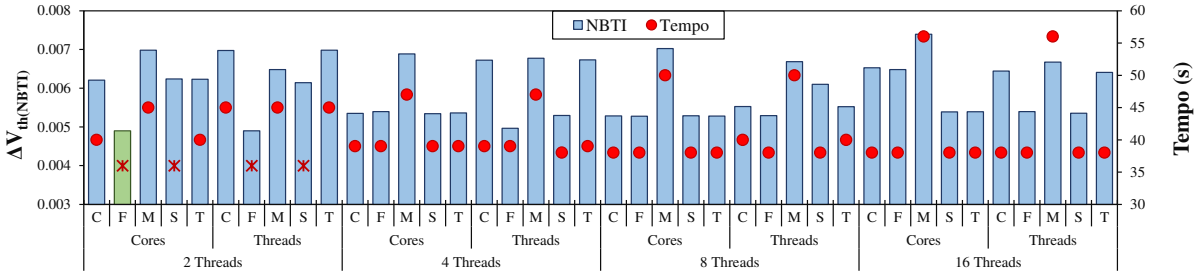
(b) Intel Xeon 32-core



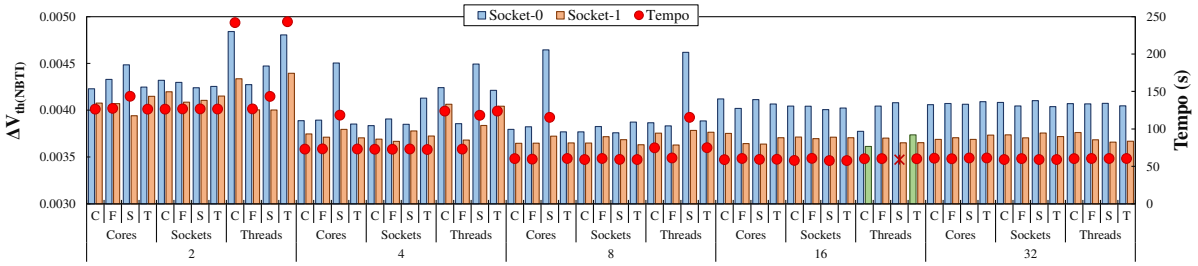
(c) IBM Power9

Fonte: O autor, 2021

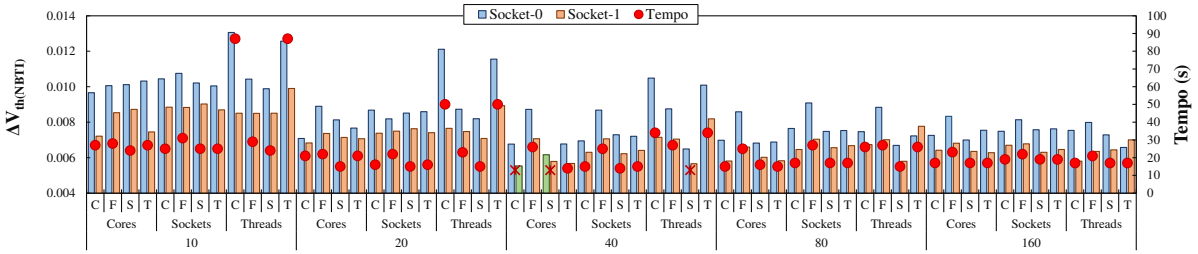
Figura 31 – Baixa taxa de comunicação: mg.C.x



(a) AMD Ryzen 7 16-core



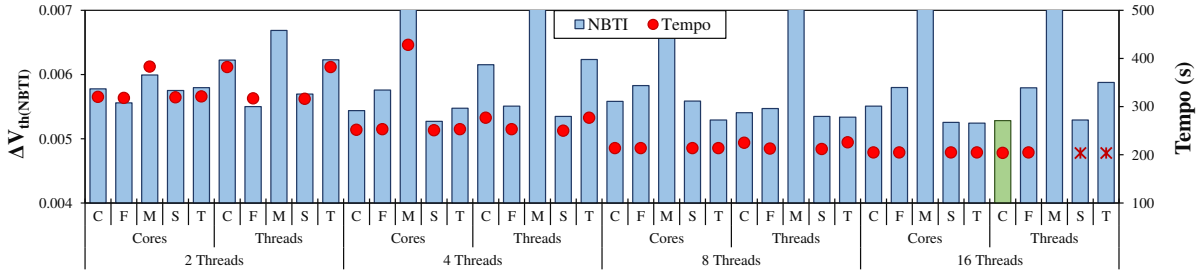
(b) Intel Xeon 32-core



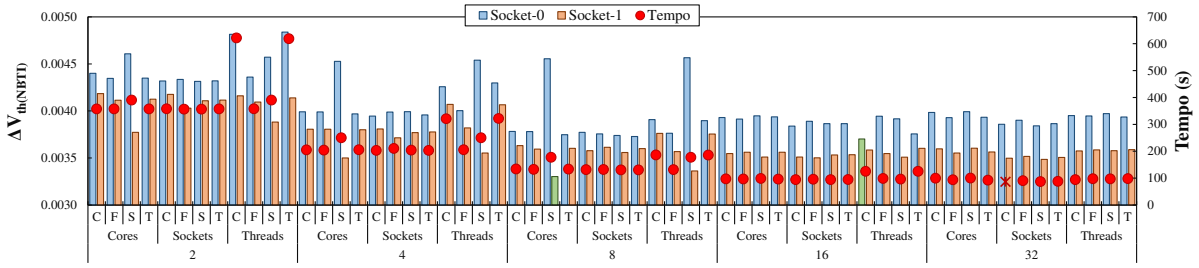
(c) IBM Power9

Fonte: O autor, 2021

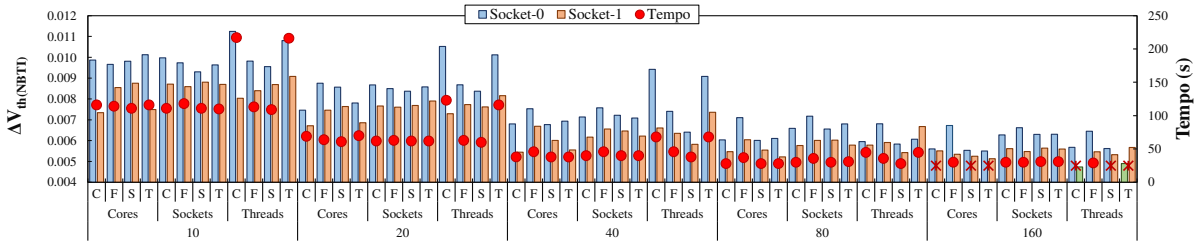
Figura 32 – Baixa taxa de comunicação: ua.C.x



(a) AMD Ryzen 7 16-core



(b) Intel Xeon 32-core



(c) IBM Power9

Fonte: O autor, 2021