Mitigando o Impacto da Degradação do Processador via Multiprogramação*

Mariana Costa[1], Sandro M. V. N. Marques[1], Fábio D. Rossi[2], Marcelo C. Luizelli[1], Antonio Carlos S. Beck[3] e Arthur F. Lorenzon[1]

[1]Universidade Federal do Pampa – Campus Alegrete – RS – Brazil

[2]Instituto Federal Farroupilha – Campus Alegrete – RS – Brasil

[3]Universidade Federal do Rio Grande do Sul – Porto Alegre – RS – Brasil

mary.costa201025@gmail.com

Abstract. *The number of cores on a single chip has been increased with each new generation of processors to meet the performance requirements of modern applications. However, the power dissipated per area has also increased, influencing the operating temperature and accelerating the phenomena responsible for the aging of processors. Hence, controlling the temperature of computational systems is essential to increase the lifespan of hardware resources. Therefore, we propose PampaAging: a dynamic, automatic, and transparent approach that adjusts the number of threads and the allocation of hardware resources for concurrent execution of a set of applications with the aim to maximize the lifetime of hardware components while also optimizing the performance of parallel applications. By the execution of twenty-four applications on two multicore architectures (Intel and AMD), we show that PampaAging can increase the lifespan of a processor by up to 42% and improve the performance by 2.52 times when compared to the common manner that parallel applications are executed.*

Resumo. *O número de núcleos em um único chip tem aumentado a cada nova geração de processadores para satisfazer a demanda de desempenho de aplicações modernas. Entretanto, a potência consumida por área também tem aumentado, influenciando a temperatura de operação e acelerando os fenômenos responsáveis pela degradação dos processadores. Neste sentido, controlar a temperatura dos sistemas computacionais é essencial para aumentar a vida útil dos recursos computacionais. Sendo assim, nós propomos PampaAging: uma abordagem dinâmica, automática e transparente que realiza o ajuste do número de threads e a alocação do recursos de hardware para execução concorrente de um conjunto de aplicações com objetivo de maximizar a vida útil dos componentes de hardware enquanto também otimiza o desempenho das aplicações paralelas. Com a execução de vinte e quatro aplicações em duas arquiteturas multicore (Intel e AMD), mostramos que PampaAging consegue melhorar em até 42% a vida útil do processador e o desempenho em 2.52 vezes em comparação à maneira padrão que aplicações paralelas são executadas.*

1. Introdução

Com o objetivo de suprir a demanda por desempenho de aplicações de diferentes domínios (e.g., aprendizado de máquina e aplicações em nuvem), o número de núcleos de processamento em sistemas computacionais de alto desempenho tem aumentado a cada geração.

*Este trabalho foi parcialmente financiado pela FAPERGS nos projetos 19/2551-0001224-1, 17/2551-0001193-7 e 19/2551-0001689-1 e PROBIC-FAPERGS

Entretanto, a potência consumida por área também tem aumentado a cada nova geração de processadores, levando a problemas significativos de dissipação de calor quando múltiplas unidades de processamento são utilizadas ao mesmo tempo. Além dos problemas comuns de resfriamento do processador, quanto maior a dissipação de calor, maior é a temperatura operacional dos componentes de *hardware*, contribuindo para a degradação e redução da sua vida útil. Portanto, controlar a temperatura operacional é essencial para evitar aumentar a vida útil destes componentes de *hardware*.

Duas principais causas de degradação em dispositivos MOS (metal-oxide-silicon) são conhecidas como NBTI (*negative bias temperature instability*) e HCI (*hot-carrier injection*). O NBTI está relacionado à geração de carga de óxido positiva e às armadilhas de interface em estruturas de MOS sob uma combinação de temperaturas elevadas e tensões negativas de portas [Stathis and Zafar 2006, White and Bernstein 2008]. Do mesmo modo, *hot-carriers* são partículas que atingem uma energia cinética muito alta ao serem aceleradas por um campo elétrico. Quando estas partículas são injetadas em regiões normalmente proibidas do dispositivo, como por exemplo, o dielétrico da porta, podem causar defeito no dispositivo. Isso, por sua vez, aumenta a tensão de limiar (V_{th}), o que tem efeitos adversos na corrente e no atraso de propagação, reduzindo o desempenho do dispositivo [Schroder and Babcock 2003]. Este aumento na tensão de limiar pode provocar um comportamento indesejado do sistema (e.g., eletromigração, ruptura dielétrica e migração de tensão [Corbetta and Fornaciari 2012]) para muitas aplicações críticas, aumentando ainda mais as despesas operacionais.

Neste sentido, diferentes técnicas em *HW*/*SW* têm sido propostas com o objetivo de controlar a temperatura do processador e, por consequência, mitigar a degradação dos componentes de *hardware*. No nível do *hardware*, o gerenciamento dinâmico de temperatura (DTM – *dynamic thermal management*) é aplicado para controlar o consumo de potência e respectiva dissipação de calor da CPU [Brooks and Martonosi 2001]. Por outro lado, as técnicas desenvolvidas no nível de *software* visam distribuir a carga de trabalho em unidades de processamento com a intenção de reduzir a criação de pontos de calor, diminuindo assim a temperatura global do processador [Medeiros et al. 2021a, Medeiros et al. 2021b]. Estas técnicas fazem uso do ajuste dinâmico do número de *threads*, da alocação de *threads* de acordo com a temperatura dos núcleos de processamento, e da redução da frequência de operação do processador através do DVFS (*dynamic voltage and frequency scaling*).

No entanto, quando o ajuste dinâmico do número de *threads* é empregado em uma aplicação com escalabilidade limitada, o sistema tende a ser sub-utilizado [Medeiros et al. 2019, Lorenzon and Beck Filho 2019]. Isto é, os recursos (i.e., núcleos e memórias *cache*) que não estiverem sendo utilizados pela aplicação ficarão ociosos. Assim, enquanto uma aplicação está sendo executada com um pequeno número de *threads*, os recursos ociosos podem ser utilizados por outra aplicação de maneira concorrente [da Silva et al. 2021]. Assim, a execução de múltiplas aplicações paralelas de forma concorrente vem sendo empregada com o objetivo principal de reduzir o tempo total ou consumo de energia da execução de um conjunto de aplicações quando comparada a execução serial de cada aplicação. No entanto, esta mesma abordagem pode também ser utilizada para melhorar o custo-benefício entre desempenho e vida útil dos componentes de *hardware*.

Portanto, considerando que o aumento da vida útil de sistemas computacionais de alto desempenho é fundamental para aumentar o número de vezes que uma aplicação (ou conjunto de aplicações) possa ser executada, nós propomos *PampaAging*. Ele é uma abordagem dinâmica e transparente para o usuário, que aplica um algoritmo de aprendizagem para (*i*) encontrar o número ideal de *threads* para uma dada aplicação paralela e (*ii*) executar diferentes aplicações paralelas ao mesmo tempo; com o objetivo de maximizar o custo-benefício entre desempenho e vida útil do processador.

Através da execução de um conjunto com vinte e quatro aplicações paralelas de diferentes domínios em duas arquiteturas *multicore* (AMD e Intel), comparamos *Pampa-*

Aging com as seguintes estratégias: *Serial*, onde as aplicações são executadas de maneira serial, uma após a outra; e *Equipart* [Creech et al. 2013a], onde os recursos computacionais são divididos de maneira igual entre as aplicações que estão sendo executadas. Mostramos que a capacidade do *PampaAging* de encontrar uma configuração ideal ou próxima do ideal para executar cada aplicação paralela de maneira concorrente é capaz de apresentar: (*i*) melhor custo-benefício entre desempenho e temperatura do processador. Isto é, *PampaAging* consegue reduzir o tempo de execução de um conjunto de aplicações, enquanto mantém a temperatura de operação do processador abaixo das demais estratégias na maior parte do tempo. (*ii*) menor impacto na degradação do processador decorrente do NBTI e HCI através da redução de temperatura e melhor uso dos recursos computacionais; (*iii*) estender a vida útil do processador em até 42% enquanto é capaz de executar 2.52 vezes mais aplicações quando comparado às estratégias apresentadas.

O restante deste artigo está estruturado como segue. Os trabalhos relacionados são apresentados na Seção 2. Na Seção 3 nós descrevemos a fundamentação teórica e o *PampaAging*. Na Seção, 4 nós descrevemos a metodologia. Então, os resultados são discutidos na Seção 5. Por fim, as conclusões são destacadas na Seção 6.

2. Trabalhos Relacionados

Nós começamos discutindo os trabalhos que propõem soluções para otimizar o uso de recursos computacionais através da multiprogramação e então descrevemos trabalhos que visam diminuir a degradação dos componentes de *hardware*. Por fim, apresentamos a contribuição do nosso trabalho perante o estado da arte.

a) Multiprogramação.[Sasaki et al. 2013] propõem C-3PO, um sistema de gerenciamento de recursos que aplica *thread packing* e DVFS para otimizar o desempenho dentro de uma restrição de potência em cargas de trabalho multiprogramadas. [Creech et al. 2013b] propõem SCAF, um sistema que ajusta em tempo de execução os recursos de *hardware* disponíveis para cada aplicação de acordo com informações em tempo real (e.g., instruções por ciclo). [Tousimojarad and Vanderbauwhede 2014] apresentam *Extended Lowest Load*, uma heurística para balancear a carga de trabalho de maneira justa entre os núcleos durante a execução de aplicações de maneira concorrente.

[Harris et al. 2014] apresentam Callisto, um gerenciador de recursos que controla a execução de aplicações OpenMP e Domino para diminuir a interferência de múltiplas aplicações executando de maneira concorrente. [Breitbart et al. 2015] avaliam o desempenho e consumo de energia de aplicações em ambiente de multiprogramação e propõem uma ferramenta para monitorar e otimizar o seu escalonamento na CPU. Na extensão do trabalho em [Breitbart et al. 2017], é apresentado um escalonador para melhorar o desempenho e consumo de energia das aplicações por meio de orquestração de máquinas virtuais e migração de *threads*. [Cho et al. 2018] apresentam NuPoCo, um *framework* para melhorar o desempenho de aplicações paralelas multiprogramadas através do ajuste dinâmico do número de *threads* para cada aplicação.

b) Redução da degradação. [Bartolini et al. 2012] propõem *self-calibrating model-predictive controller*, uma ferramenta que aplica gerenciamento de temperatura e frequência em arquiteturas *multicore*. [Cho et al. 2012] apresentam um método de migração de potência espaço-temporal que usa a ativação/desativação dos núcleos do processador para diminuir o impacto da temperatura na degradação.

[Rahimi et al. 2013] propõem uma estratégia de realocação da carga de trabalho da CPU para reduzir a degradação em arquiteturas GPGPU. [Khdr et al. 2014] propõem uma técnica de gerenciamento dinâmico de temperatura que aplica migração de *threads* e DVFS para diminuir a degradação do processador. [Khdr et al. 2018] apresentam *Guard-Boost*, uma técnica de *boosting* baseada em DVFS para reduzir o efeito da degradação do processador em curto e longo prazo sem impactar no desempenho das aplicações. [Sharifi et al. 2020] propõem uma técnica para diminuir o estresse dos núcleos do processador por meio de classificação das aplicações e migração de *threads*.

c) Nossas Contribuições. Considerando os trabalhos destacados no item **(a)**, embora as estratégias apliquem multiprogramação, o objetivo não consiste em otimizar o desempenho das aplicações ao mesmo tempo em que se avalia os efeitos de NBTI e HCI na degradação do processador. Por outro lado, os trabalhos que focam em reduzir a degradação dos componentes de *hardware* (item **b**), não empregam a execução concorrente de aplicações para melhorar o uso dos recursos computacionais. Portanto, diferentemente dos trabalhos discutidos, este artigo propõe uma abordagem para maximizar o número de vezes que um conjunto de aplicações possa ser executada até o fim da vida útil de um dado processador levando em consideração diferentes *governors* DVFS. *PampaAging* é dinâmico, automático e transparente, não exigindo nenhuma modificação de código ou recompilação por parte do usuário.

3. Mitigando o Impacto da Degradação do Processador

PampaAging tem como objetivo principal maximizar o número de vezes que aplicações (ou conjunto de aplicações) possam ser executadas até o fim da vida útil de sistemas computacionais (e.g., HPC, computação em nuvem) através da escolha do número ideal de *threads* para cada aplicação e da alocação de várias aplicações ao mesmo tempo. Para tanto, a métrica de otimização avaliada por *PampaAging* considera o custo-benefício entre desempenho (o tempo de execução) e a degradação total por conta do NBTI e HCI.

O fluxo de execução aplicado por *PampaAging* é definido como segue: (*i*) O usuário fornece como entrada para *PampaAging* uma aplicação ou uma lista com aplicações e seus conjuntos de entrada que serão executadas na arquitetura alvo. (*ii*) Se uma aplicação ainda não foi treinada, *PampaAging* aplica um algoritmo de aprendizagem (discutido abaixo) sobre a execução da aplicação com uma entrada representando 10% da entrada padrão para encontrar o melhor número de *threads* para executar a respectiva aplicação que otimiza a métrica objetivo. (*iii*) Uma vez que a configuração é encontrada, ela é armazenada no banco de dados implementado pelo *PampaAging* com o objetivo de reduzir o tempo de aprendizagem caso a aplicação seja executada mais de uma vez na arquitetura. Caso contrário, *PampaAging* acessa a base de dados para obter a melhor configuração para a aplicação que já foi treinada. (*iv*) Com a lista de aplicações já treinadas, *PampaAging* emprega um algoritmo de alocação das aplicações nos recursos (e.g., núcleos de processamento) disponíveis. (*v*) Por fim, *PampaAging* verifica continuamente se a lista de aplicações contém uma nova aplicação para executar. Esta etapa proporciona dinamicidade, uma vez que aplicações podem ser inseridas sob demanda para execução na arquitetura alvo.

3.1. Modelando a Degradação dos Componentes de *Hardware*

NBTI é um fenômeno de envelhecimento que degrada as características elétricas dos transistores *pMOS* e *nMOS*. Ele aumenta a tensão limite (V_{th}) ao longo da vida útil do processador, o que geralmente torna o chaveamento dos transistores mais lento [Corbetta and Fornaciari 2012]. Do mesmo modo, *hot-carriers* são partículas que atingem uma energia cinética muito alta ao serem aceleradas por um campo elétrico. Quando estas partículas são injetadas em regiões normalmente proibidas do dispositivo, como por exemplo, o dielétrico do *gate*, podem causar defeito no dispositivo. Isso afetará o atraso no caminho crítico do processador, gerando violações de tempo e erros. Ultimamente, os dispositivos têm uma vida útil até o momento que o tempo do caminho crítico torna-se maior do que o período do relógio e, portanto, os erros de tempo que aparecem impedem a geração de cálculos confiáveis.

Nós utilizamos o modelo de degradação (e.g., *aging*) proposto pelos autores em [Lee et al. 2018, Oboril and Tahoori 2012], que considera o atraso total decorrente do NBTI e HCI. Dada a temperatura do processador (T, medida em *Kelvin*), a tensão de operação (V_{dd}), o período de medição (t_m), o $\Delta V_{th(NBTI)}$ no instante de tempo $t > 0$ pode ser estimado pela Equação 1 [1]. Por outro lado, o $\Delta V_{th(HCI)}$ pode ser estimado

[1]Valores das constantes foram obtidas de [Oboril and Tahoori 2012, Bhardwaj et al. 2006,

através da Equação 6. Por fim, o $\Delta V_{th(Total)}$ é a soma dos valores obtidos em $\Delta V_{th(NBTI)}$ e $\Delta V_{th(HCI)}$ Para calcular o $\Delta V_{th(Total)}$ de cada núcleo de processamento (Equação 1) para o processador Intel, nós usamos o comando *Linux sensors* para obter a temperatura atual da CPU e a tensão de operação por segundo. Para o processador AMD, nós usamos a ferramenta MatEx [Pagani et al. 2015], onde a temperatura de cada núcleo de processamento é calculada com base no *floorplan* da arquitetura e as medições de potência são obtidas com a biblioteca APM. Por fim, para calcular o *duty cycle* de cada núcleo executando a aplicação (δ_C), nós consideramos a razão de tempo que cada núcleo de processamento está sob stress.

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

com,

$$v(T) = \xi_4 \cdot exp\left(-\frac{E_a}{kT}\right); \tag{2}$$

$$u(V_{dd}) = (V_{dd} - V_{th}) \cdot exp\left(\frac{V_{dd} - V_{th}}{E_0}\right); \tag{3}$$

$$w = 1 - \left(1 - \frac{\xi_1 + \sqrt{\xi_3 \cdot v(T) \cdot (1 - \delta(t)) \cdot t_m}}{\xi_2 + \sqrt{v(T) \cdot t}}\right)^{\frac{1}{2n}} \tag{4}$$

$$\delta_C = \frac{cyc_{stress}}{cyc_{total}} \tag{5}$$

$$\Delta V_{th(HCI)} = A_H \cdot \sqrt{\alpha_{avg,C}} \cdot u(V_{dd}) \cdot v(T) \cdot \sqrt{\alpha_C \cdot f \cdot t}; \tag{6}$$

com,

$$u(V_{dd}) = exp(\frac{V_{dd} - V_{th}}{E_1}) \tag{7}$$

$$v(T) = exp(\frac{-E_a}{kT}) \tag{8}$$

3.2. Algoritmo de Aprendizagem

PampaAging recebe do usuário uma lista L com m aplicações paralelas $P = \{P_1, P_2, \ldots P_m\}$. Cada aplicação paralela em P pode ser executada por no máximo c núcleos de processamento distintos $C = \{C_1, C_2, \ldots, C_n\}$, onde *n* é o número máximo de *threads* de *hardware* disponível na arquitetura alvo. Nós denotamos por $L(p)$ o conjunto de aplicações na lista L e o problema de otimização consiste em atribuir aplicações paralelas a um conjunto de núcleos de processamento. Nós denotamos por C^* o conjunto de todos os subconjuntos de threads/núcleos em C, isto é, $C^* = \cup_{j=1}^{|C|} \binom{C}{j}$. Uma atribuição viável pode ser definida como uma função de $\phi : P_i \rightarrow C^* (\forall P_i \in P)$ que atribui um subconjunto de threads/núcleos para uma aplicação $P_i \in P$. Portanto, $M : (P \times C^*) \rightarrow R^+$ corresponde a métrica objetivo (e.g., tradeoff entre desempenho e degradação) da execução da aplicação P_i em *c* núcleos de processamento. Por fim, consideramos que ϕ está otimizado se $\sum_{P_i \in P} M(P_i, \phi(P_i))$ é máximo.

O algoritmo de aprendizagem empregado pelo *PampaAging* está descrito em Algoritmo 1. Ele recebe o conjunto de núcleos de processamento *C*; a lista de aplicações

Medeiros et al. 2021a, Amrouch et al. 2014]: $n = 1/6$, $E_0 = 0.335$, $E_a = 0.49$, $C = 0.0163$, $\xi_1 = 0.9$, $\xi_2 = 0.5$, $\xi_3 = 1.0$, $\xi_4 = 10^{-8}$, $r = 1.6$, e A_N de acordo com [Amrouch et al. 2014], e *k* é a constante de *Boltzmann*

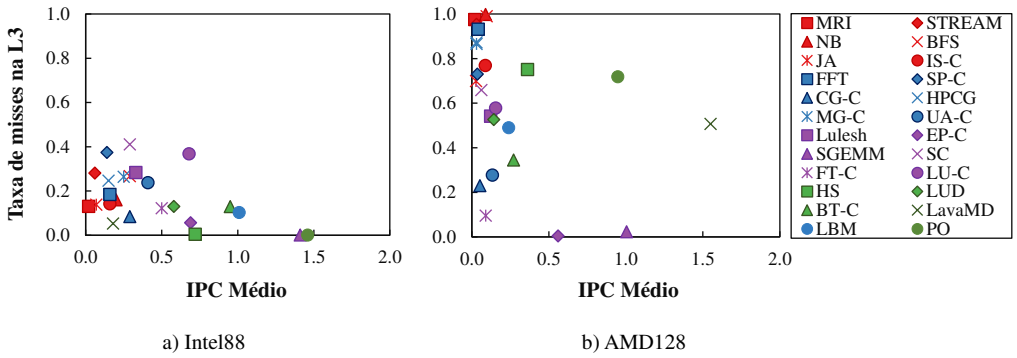
Algorithm 1 Algoritmo de Aprendizagem

```
Input: C ← {C_1, C_2, ..., C_k}: conjunto de threads/núcleos
       L: Lista de aplicações
       ω: Número inicial de threads
       β: Fator de aumento no número de threads
1:
2:  function PAMPAAGING(...)
3:      while true do
4:          for cada aplicação m em L do
5:              busca_TLP(m)
6:          end for
7:          aloca_Apps(L)
8:          monitora lista de apps para treinamento
9:      end while
10: end function
11:
12:
13: function BUSCA_TLP(m)
14:     φ(ω, m) ← ∞                          ▷ Melhor número de threads até o momento
15:     φ' ← 0                               ▷ máximo valor da métrica encontrado até o momento
16:     Ω ← ω e M' ← M(Ω, m)
17:     while M' ≥ φ' do
18:         φ' ← M' and Ω ← Ω + β and M' ← M(Ω, m)
19:     end while
20:     if φ' ≥ φ(Ω, m) then
21:         φ(Ω, m) ← φ'
22:     end if
23:     return φ(Ω, m)
24: end function
25:
26:
27: function ALOCA_APPS(L)
28:     while Lista L não é vazia do
29:         while Existem recursos de hardware disponíveis do
30:             for cada aplicação m em L do
31:                 Executa aplicação m
32:             end for
33:         end while
34:         while Não existem recursos de hardware disponíveis do
35:             Monitora execução das aplicações
36:         end while
37:     end while
38: end function
```

L, que consiste de um arquivo (*buffer*) com as aplicações que serão executadas; e dois parâmetros: ω – o número inicial de *threads* dado para uma aplicação *A*, e β – o fator de incremento para o número de *threads* dado para *A* (nos experimentos, consideramos $\omega = 2$ e $\beta = 2$). Enquanto *PampaAging* estiver em execução, (linhas 3 – 9), duas funções principais são executadas: *busca_TLP* e *aloca_Apps*. Elas são descritas em detalhe abaixo.

Função *busca_TLP*. Para cada aplicação m da lista L, é aplicado o algoritmo de busca pelo número ideal de *threads* sob a execução da aplicação com uma entrada correspondente a 10% da entrada padrão. Conforme mostrado em [Berned et al. 2021], realizar o algoritmo de busca sobre uma entrada representando aproximadamente 10% da entrada padrão é suficiente para uma convergência próxima do número ideal de *threads* na maioria dos casos. Esta função inicia a execução da aplicação m com o número de *threads* igual a Ω. Este número de *threads* é incrementado pelo fator β enquanto maximiza-se o valor da função avaliação $M(\phi) = M'$. Quando o valor da função avaliação não é melhorado, retorna-se o número de *threads* onde a aplicação parou de escalar. Considerando que encontrar o número ideal de *threads* para executar uma aplicação paralela pode ser definido como um problema convexo de otimização, existirá apenas um número específico de *threads* que entregará o melhor *trade-off* entre desempenho e degradação.

Função *aloca_Apps*. Uma vez que o algoritmo de aprendizagem foi empregado para as aplicações da lista L, esta função é responsável por otimizar a utilização dos



a) Intel88 b) AMD128

Figura 1. Características de cada aplicação em cada arquitetura *multicore*

recursos de *hardware* através da alocação de múltiplas aplicações paralelas de maneira concorrente. Para tanto, uma aplicação será alocada quando satisfazer os seguintes requisitos: (*i*) quando houver núcleos de processamento disponível (i.e., número de *threads* da aplicação é menor ou igual ao número de núcleos disponíveis); e (*ii*) quando houver memória disponível. Portanto, a função recebe a lista de aplicações L enquanto existir aplicações para serem executadas (linha 28), o algoritmo verifica os recursos de hardware (núcleos de processamento e memória disponível). Se existir recurso disponível, encontra uma aplicação m na lista L que satisfaz os requisitos para execução. Caso não tenha recurso disponível, o algoritmo monitora a execução até existir recursos disponíveis para iniciar a execução de novas aplicações. Uma vez que a lista de aplicações foi executada, *PampaAging* verifica se existem novas aplicações para execução (linha 8).

4. Metodologia

Benchmarks. Para a realização dos experimentos, vinte e quatro aplicações foram utilizadas com as respectivas entradas. **Nove kernels, pseudo-aplicações e benchmarks do NAS Parallel Benchmark** [Bailey et al. 1991]: *BT-C*, *CG-C*, *MG-C*, *EP-C*, *FT-C*, *IS-C*, *LU-C*, *SP-C* e *UA-C*. **Quatro aplicações da suíte de benchmarks do Rodinia** [Che et al. 2009], com a entrada padrão: *Hotspot*, *LavaMD*, *LUD* e *Streamcluster*. **Quatro aplicações da suíte de benchmark do Parboil** [Stratton et al. 2012]: *Sgemm-medium*, *Lbm-long*, *Mri-gridding-small* e *Bfs-SF*. **Sete aplicações de diferentes domínios**, com a entrada padrão: *FFT*, *HPCCG*, *Lulesh 2.0*, *Nbody*, *Método de Jacobi*, *Poisson*, e *Stream*. As aplicações escolhidas possuem diferentes comportamentos com relação ao acesso à memória e utilização do processador. Neste sentido, nós caracterizamos as aplicações de acordo com a taxa de *misses* na cache L3 e o IPC médio, conforme mostrado na Figura 1. Os dados para esta classificação foram retirados diretamente dos contadores de *hardware* através do Intel *Performance Counter Monitor* e do *AMDuProf*. Para tanto, cada aplicação foi executada com o número de *threads* igual ao número de núcleos disponíveis em cada arquitetura.

Ambiente de Execução. Os experimentos foram realizados em duas arquiteturas multicores: *Intel88*, um sistema bi-processado Intel Xeon E5-2699 v4 Broadwell, com 44 núcleos (88 threads), 110MB de memória cache L3 e 256GB de memória RAM[2]; e *AMD128*, AMD Ryzen Threadripper 3990X com 64 núcleos (128 threads via SMT), 256MB de memória cache L3 e 32GB de memória RAM. Utilizamos o Sistema Operacional Ubuntu 20.04 com o kernel 5.11.0-25 e as aplicações foram compiladas com GCC/G++ 10.2.0 e *flag* de otimização *-O3*. Por fim, para calcular o impacto do NBTI e HCI na degradação do processador (Seção 3), os dados foram extraídos diretamente do *hardware* através de ferramentas do Sistema Operacional Linux. A temperatura e tensão

[2]Alguns experimentos deste trabalho utilizaram os recursos da infraestrutura PCAD, http://gppd-hpc.inf.ufrgs.br, no INF/UFRGS

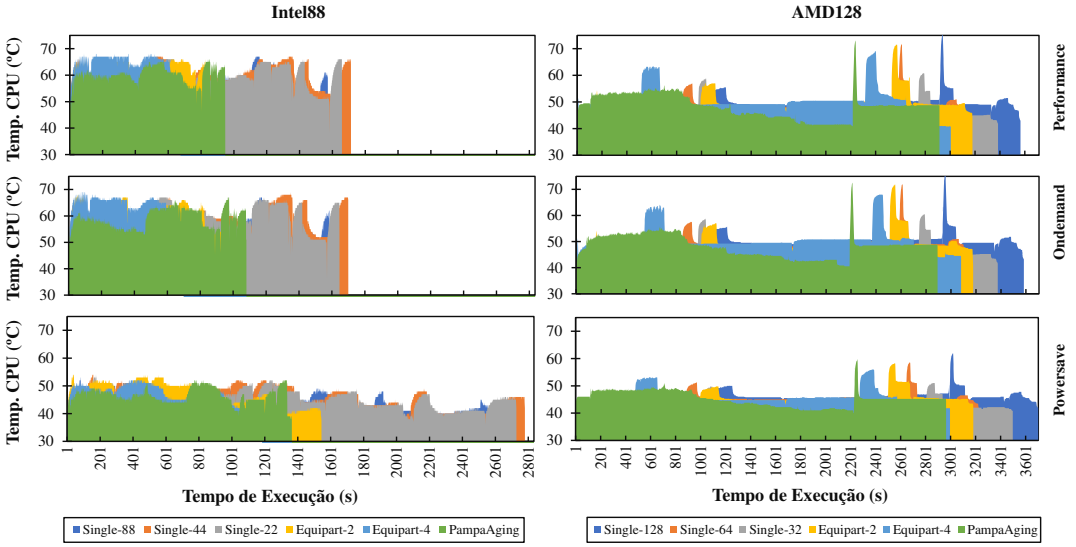


Figura 2. Temperatura ao longo do tempo para cada uma das configurações e arquiteturas

de operação de cada processador foi capturada através da ferramenta de monitoramento *lm_sensors*, enquanto que a frequência de operação foi obtida através do *cpu-freq utils*.

5. Resultados Experimentais

Nós comparamos a execução de *PampaAging* sobre a lista de aplicações descritas na Seção 4 com os seguintes modos de execução:

- **Serial**, na qual as aplicações foram executadas de maneira serial (i.e., apenas uma aplicação por vez). Neste modo de execução, consideramos diferentes configurações de acordo com o número de *threads* (NT) atribuídos a cada aplicação com o intuito de observar o comportamento de aplicações com escalabilidade limitada: *Serial-NT*, com o número total de *threads* atribuídos à aplicação; *Serial-NT/2*, onde cada aplicação foi executada com *NT/2 threads*; e *Serial-NT/4*, onde cada aplicação foi executada serialmente com *NT/4 threads*.
- **Equipart-2**, uma abordagem proposta por [Creech et al. 2013a], onde duas aplicações foram executadas ao mesmo tempo, cada uma com metade dos recursos de *hardware*. Nós também consideramos uma variação desta estratégia, chamada de *Equipart-4*, onde quatro aplicações são executadas de maneira concorrente, cada uma utilizando um quarto dos recursos de *hardware*.

Adicionalmente, para avaliar a eficiência de *PampaAging* quando diferentes configurações de DVFS são aplicadas, ele foi avaliado com diferentes *governors* DVFS: *powersave*, no qual a frequência do processador é configurada para o mínimo possível; *performance*, onde a frequência dos núcleos é configurada para o máximo possível; e *ondemand*, onde a frequência é ajustada de acordo com a carga de trabalho de cada CPU. Por fim, para reduzir o efeito das técnicas de *boosting* (i.e., Intel Turbo Boost e AMD Turbo Core), estas foram desativadas.

Inicialmente, vamos discutir o impacto das estratégias no comportamento da temperatura da CPU ao executar todo o conjunto de aplicações descritos na Seção 4. Conforme apresentado na Seção 3, a temperatura de operação tem um fator primordial na degradação dos componentes de *hardware*. Portanto, quanto menor for a temperatura, melhor. Para tanto, a Figura 2 ilustra, a temperatura (*eixo y*) por segundo de execução (*eixo x*) das aplicações em cada arquitetura e *governor* DVFS. A primeira observação que pode ser feita é que *PampaAging* apresentou menor tempo de execução que as demais

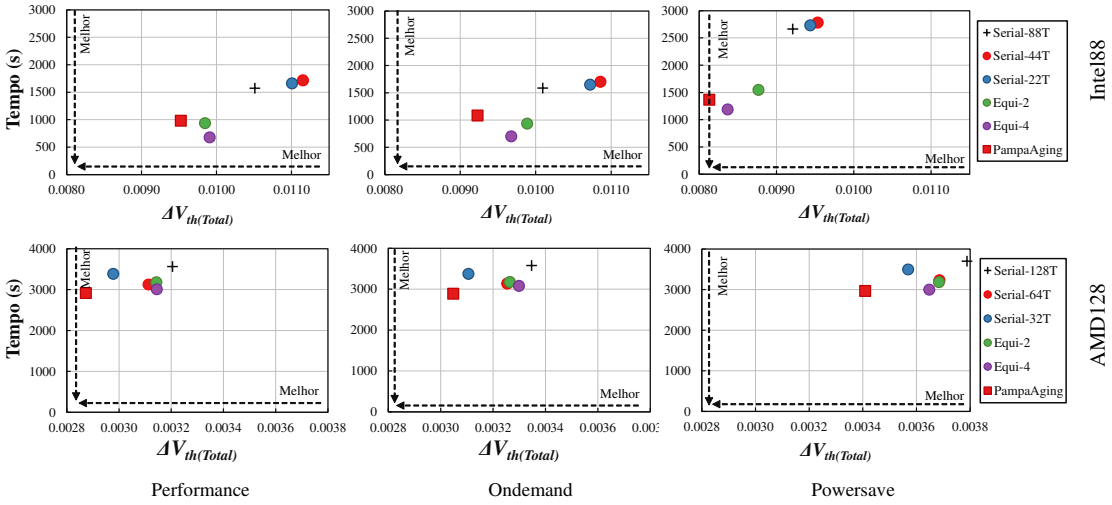


Figura 3. Tempo de Execução e $\Delta V_{th(Total)}$ para cada modo de execução, arquitetura e *governor* DVFS.

abordagens, devido (*i*) a capacidade de convergir para um número de *threads* ideal ou próximo do ideal para cada aplicação e (*ii*) a alocação de aplicações de modo que o uso dos recursos computacionais seja otimizado. Adicionalmente, pode-se observar que, durante a maior parte da execução das aplicações, a temperatura do processador foi menor quando *PampaAging* foi utilizado para gerenciar a execução.

Uma vez que *PampaAging* foi capaz de reduzir o tempo de execução e ao mesmo tempo manter os níveis de temperatura de operação do processador menor do que as demais estratégias, o principal efeito é a redução do $\Delta V_{th(Total)}$ em decorrência do NBTI e HCI (Ver Seção 3.2). Nós ilustramos este cenário na Figura 3. Ela mostra o tempo de execução total(*eixo x*) e o $\Delta V_{th(Total)}$ (*eixo y*) para o conjunto de aplicações em cada arquitetura e *governor* DVFS. Quanto mais próximo da origem (ponto 0-0) a configuração estiver, melhor é o custo-benefício entre desempenho e impacto do NBTI e HCI na degradação dos componentes de *hardware*. Neste sentido, independente da arquitetura e governor *DVFS*, *PampaAging* apresentou melhores resultados. As principais razões para tal comportamento são as seguintes: (*i*) *PampaAging* otimiza o número de *threads* ativas para cada aplicação, o que tem impacto direto no NBTI através da variável δ_c. Por outro lado, as demais abordagens têm número fixo de *threads* para cada aplicação, fazendo mau uso dos recursos computacionais, na maioria dos casos. (*ii*) Por utilizar os recursos de maneira próxima do ideal, não há subutilização dos recursos computacionais, nem criação de pontos isolados de calor.

O resultado de ser capaz de otimizar o desempenho e ao mesmo tempo manter a temperatura de operação do processador em níveis mais baixos pela maior parte da execução das aplicações é maximizar a vida útil do processador ao mesmo tempo que aumenta a quantidade de vezes que um conjunto de aplicações poderá ser executado dentro deste limite. A Figura 4 apresenta este cenário para cada arquitetura e *governor* DVFS. Nela, o *eixo x* apresenta o número de vezes que o conjunto de aplicações pode ser executado até o fim da vida útil do processador, enquanto que o *eixo y* destaca o respectivo tempo de vida útil. Nós consideramos que o fim da vida útil de um processador ocorre quando V_{th} aumenta em 10% [Lee et al. 2018, Oboril and Tahoori 2012]. Como um exemplo, a configuração *Serial-88T* executando no processador *Intel88* com o *governor performance* será capaz de completar 70×10^3 execuções antes do fim da sua vida útil, estimada de 3.4 anos.

Conforme podemos observar na Figura 4, *PampaAging* entrega o melhor custo benefício entre desempenho (número de vezes que as aplicações poderão ser executadas)

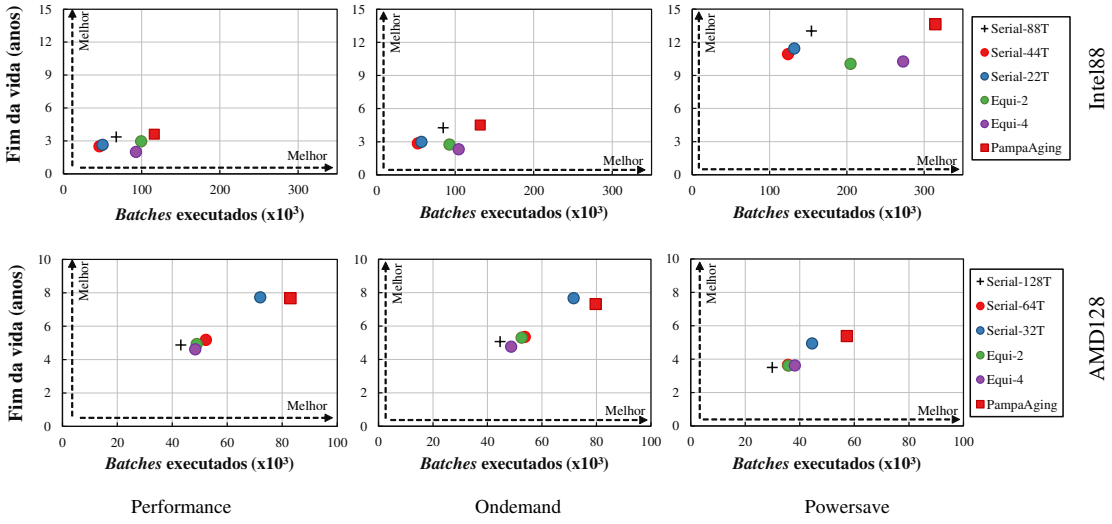


Figura 4. Número de vezes que o conjunto de aplicações poderá ser executado até o fim da vida útil de cada processador para cada estratégia e *governor* DVFS.

e degradação do processador (estimado fim da vida útil) independente da arquitetura e do governor *DVFS*. Quando o objetivo do sistema é desempenho (*governor performance*), *PampaAging* não só é capaz de apresentar melhor desempenho (executar mais aplicações), mas também apresentar maior tempo de vida útil. Por outro lado, quando o objetivo do *governor* é reduzir o consumo de *potência* (*powersave*), *PampaAging* também apresentou melhores resultados. Quando consideramos a média de todos os *governors* e estratégias em cada arquitetura, no caso mais significativo, *PampaAging* é capaz de executar 2.52 vezes mais aplicações e estender a vida útil em 42.2% anos que a estratégia *Serial-44T* no processador *Intel88*; e executar 1.87 vezes mais aplicações com uma vida útil 52% maior que a configuração *Serial-64T* no processador *AMD128*. Quando comparado com as demais configurações, *PampaAging* apresenta resultados melhores, porém, com diferentes taxas de melhorias.

6. Conclusão

Este artigo apresentou *PampaAging*, uma abordagem capaz de (*i*) encontrar o número ideal ou próximo do ideal de threads para executar cada aplicação paralela através de um algoritmo dinâmico; e (*ii*) alocar automaticamente e de maneira transparente a execução de aplicações paralelas de forma concorrente com o objetivo de maximizar o desempenho e tempo de vida útil do processador. *PampaAging* é transparente, automático e dinâmico, não exigindo modificações no código ou recompilação por parte do usuário. Como trabalhos futuros, nós iremos adaptar o algoritmo implementado por *PampaAging* para arquiteturas heterogêneas.

Referências

Amrouch, H., van Santen, V. M., Ebi, T., Wenzel, V., and Henkel, J. (2014). Towards interdependencies of aging mechanisms. In *IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pages 478–485.

Bailey, D. H., Barszcz, E., Barton, J. T., Browning, D. S., Carter, R. L., Dagum, L., Fatoohi, R. A., Frederickson, P. O., Lasinski, T. A., Schreiber, R. S., Simon, H. D., Venkatakrishnan, V., and Weeratunga, S. K. (1991). The nas parallel benchmarks & summary and preliminary results. In *ACM/IEEE SC*, pages 158–165, USA. ACM.

Bartolini, A., Cacciari, M., Tilli, A., and Benini, L. (2012). Thermal and energy management of high-performance multicores: Distributed and self-calibrating model-

predictive controller. *IEEE Transactions on Parallel and Distributed Systems*, 24(1):170–183.

Berned, G., Rossi, F. D., Luizelli, M. C., de Souza, S. X., Beck, A. C. S., and Lorenzon, A. F. (2021). Low learning-cost offline strategies for EDP optimization of parallel applications. *J. Syst. Archit.*, 114:101959.

Bhardwaj, S., Wang, W., Vattikonda, R., Cao, Y., and Vrudhula, S. (2006). Predictive modeling of the nbti effect for reliable design. In *IEEE Custom Integrated Circuits Conference 2006*, pages 189–192.

Breitbart, J., Pickartz, S., Lankes, S., Weidendorfer, J., and Monti, A. (2017). Dynamic co-scheduling driven by main memory bandwidth utilization. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, pages 400–409. IEEE.

Breitbart, J., Weidendorfer, J., and Trinitis, C. (2015). Case study on co-scheduling for hpc applications. In *2015 44th International Conference on Parallel Processing Workshops*, pages 277–285. IEEE.

Brooks, D. and Martonosi, M. (2001). Dynamic thermal management for high-performance microprocessors. In *Proceedings HPCA Seventh International Symposium on High-Performance Computer Architecture*, pages 171–182.

Che, S., Boyer, M., Meng, J., Tarjan, D., Sheaffer, J. W., Lee, S.-H., and Skadron, K. (2009). Rodinia: A benchmark suite for heterogeneous computing. In *IEEE Int. Symp. on Workload Characterization*, pages 44–54, DC, USA. IEEE Computer Society.

Cho, M., Kersey, C., Gupta, M. P., Sathe, N., Kumar, S., Yalamanchili, S., and Mukhopadhyay, S. (2012). Power multiplexing for thermal field management in many-core processors. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 3(1):94–104.

Cho, Y., Guzman, C. A. C., and Egger, B. (2018). Maximizing system utilization via parallelism management for co-located parallel applications. In *Proceedings of the 27th International Conference on Parallel Architectures and Compilation Techniques*, pages 1–14.

Corbetta, S. and Fornaciari, W. (2012). Nbti mitigation in microprocessor designs. In *Proceedings of the Great Lakes Symposium on VLSI*, GLSVLSI '12, page 33–38, New York, NY, USA. Association for Computing Machinery.

Creech, T., Kotha, A., and Barua, R. (2013a). Efficient multiprogramming for multicores with scaf. In *46th Annual IEEE/ACM Int. Symp. on Microarchitecture*, MICRO-46, page 334–345, New York, NY, USA. ACM.

Creech, T., Kotha, A., and Barua, R. (2013b). Efficient multiprogramming for multicores with scaf. In *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 334–345. IEEE.

da Silva, V. S., Nogueira, A. G., de Lima, E. C., de A. Rocha, H. M., Serpa, M. S., Luizelli, M. C., Rossi, F. D., Navaux, P. O., Beck, A. C. S., and Francisco Lorenzon, A. (2021). Smart resource allocation of concurrent execution of parallel applications. *Concurrency and Computation: Practice and Experience*, page e6600.

Harris, T., Maas, M., and Marathe, V. J. (2014). Callisto: Co-scheduling parallel runtime systems. In *Proceedings of the Ninth European Conference on Computer Systems*, pages 1–14.

Khdr, H., Amrouch, H., and Henkel, J. (2018). Aging-aware boosting. *IEEE Transactions on Computers*, 67(9):1217–1230.

Khdr, H., Ebi, T., Shafique, M., and Amrouch, H. (2014). mdtm: Multi-objective dynamic thermal management for on-chip systems. In *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. IEEE.

Lee, H., Shafique, M., and Al Faruque, M. A. (2018). Aging-aware workload management on embedded gpu under process variation. *IEEE Transactions on Computers*, 67(7):920–933.

Lorenzon, A. F. and Beck Filho, A. C. S. (2019). *Parallel computing hits the power wall: principles, challenges, and a survey of solutions*. Springer Nature.

Medeiros, T. S., Berned, G. P., Navarro, A., Rossi, F. D., Luizelli, M. C., Brandalero, M., Hübner, M., Beck, A. C. S., and Lorenzon, A. F. (2021a). Aging-aware parallel execution. *IEEE Embedded Systems Letters*, 13(3):122–125.

Medeiros, T. S., Pereira, L., Rossi, F. D., Luizelli, M. C., Beck, A. C. S., and Lorenzon, A. F. (2019). Transparent aging-aware thread throttling. In *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 1–8.

Medeiros, T. S., Pereira, L., Rossi, F. D., Luizelli, M. C., Beck, A. C. S., and Lorenzon, A. F. (2021b). Mitigating the processor aging through dynamic concurrency throttling. *Journal of Parallel and Distributed Computing*.

Oboril, F. and Tahoori, M. B. (2012). Extratime: Modeling and analysis of wearout due to transistor aging at microarchitecture-level. In *IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, pages 1–12.

Pagani, S., Chen, J., Shafique, M., and Henkel, J. (2015). Matex: Efficient transient and peak temperature computation for compact thermal models. In *DATE*, pages 1515–1520.

Rahimi, A., Benini, L., and Gupta, R. K. (2013). Aging-aware compiler-directed vliw assignment for gpgpu architectures. In *2013 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE.

Sasaki, H., Imamura, S., and Inoue, K. (2013). Coordinated power-performance optimization in manycores. In *Proceedings of the 22nd international conference on Parallel architectures and compilation techniques*, pages 51–61. IEEE.

Schroder, D. K. and Babcock, J. A. (2003). Negative bias temperature instability: Road to cross in deep submicron silicon semiconductor manufacturing. *Journal of applied Physics*, 94(1):1–18.

Sharifi, F., Rohbani, N., and Hessabi, S. (2020). Aging-aware context switching in multicore processors based on workload classification. *IEEE Computer Architecture Letters*, 19(2):159–162.

Stathis, J. H. and Zafar, S. (2006). The negative bias temperature instability in mos devices: A review. *Microelectronics Reliability*, 46(2-4):270–286.

Stratton, J. A., Rodrigues, C., Sung, I.-J., Obeid, N., Chang, L.-W., Anssari, N., Liu, G. D., and Hwu, W.-m. W. (2012). Parboil: A revised benchmark suite for scientific and commercial throughput computing. *Center for Reliable and High-Performance Computing*, 127.

Tousimojarad, A. and Vanderbauwhede, W. (2014). An efficient thread mapping strategy for multiprogramming on manycore processors. *Parallel Computing: Accelerating Computational Science and Engineering (CSE), Advances in Parallel Computing*, 25:63–71.

White, M. and Bernstein, J. B. (2008). Microelectronics reliability : physics-of-failure based modeling and lifetime evaluation. Technical Report JPL Publication 08-5 2/08, National Aeronautics and Space Administration, Jet Propulsion Laboratory, Pasadena, California.