Explorando a Variabilidade de Processo para Otimizar a Eficiência Energética em Servidores de Nuvem

Thiago dos S. Gonçalves[1], **Antonio Carlos S. Beck**[1] **e Arthur F. Lorenzon**[1]

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

{thiago.goncalves, caco, aflorenzon}@inf.ufrgs.br

Resumo. *O número crescente de núcleos em um único chip permite que servidores em nuvem mais poderosos explorem melhor o paralelismo no nível de requisições. No entanto, isso também leva a problemas imprevistos de potência, que podem acelerar o envelhecimento dos componentes de hardware e causar erros suaves ou até mesmo falhas. Portanto, o gerenciamento inteligente da dissipação de potência tornou-se crítico e imprevisível devido à variabilidade inerente do processo - uma vez que a potência máxima varia entre os núcleos, independentemente de operarem na mesma frequência operacional. Com base nisso, propomos PowerSaver, uma abordagem para otimizar a eficiência energética dos servidores em nuvem. PowerSaver distribui automaticamente a carga de trabalho entre os núcleos e aplica ajuste dinâmico de frequência dos domínios de core e uncore com base no comportamento das tarefas e na carga dos sistemas para reduzir a potência dissipada. Resultados em quatro arquiteturas multicore idênticas mostram que PowerSaver reduz a potência de pico em até 25.6% em relação ao escalonador padrão do Sistema Operacional Linux e em até 20.3% comparado ao HiMap, uma estratégia do estado da arte, com mínimo impacto no desempenho.*

1. Introdução

A crescente demanda por *software-as-a-service* (SaaS) tem aumentado o foco nas infraestruturas de *warehouses* para possibilitar uma integração mais ampla com serviços avançados na nuvem. Esses serviços abrangem uma grande variedade de aplicações que abrangem diversas áreas, como *machine learning*, pesquisa biomédica, e processamento de vídeo/áudio. Dentro desses ecossistemas em nuvem, as cargas de trabalho exibem um alto grau de heterogeneidade, frequentemente resultante das diversas solicitações dos usuários. Portanto, um dos principais desafios consiste em oferecer serviços com latência mínima, ao mesmo tempo em que otimiza o uso dos recursos e emprega de forma inteligente o paralelismo no nível de requisições (*Request Level Parallelism* - RLP).

Os avanços na tecnologia de transistores permitiram aumentar o número de núcleos em um único *chip* para atender às crescentes demandas de desempenho. No entanto, o fim do conhecido *Dennard Scaling* [Bohr 2007] se tornou evidente, resultando em aumento na dissipação de potência por unidade de área em cada nova geração de nodo computacional [Lorenzon and Beck Filho 2019]. Consequentemente, isso teve um impacto significativo na temperatura operacional dos componentes de *hardware*, trazendo consigo desafios consideráveis. Isso exige uma infraestrutura de resfriamento cada vez mais cara e complexa, acelerando o processo de degradação dos componentes de *hardware* e resultando em comportamentos prejudiciais do sistema, como uma maior

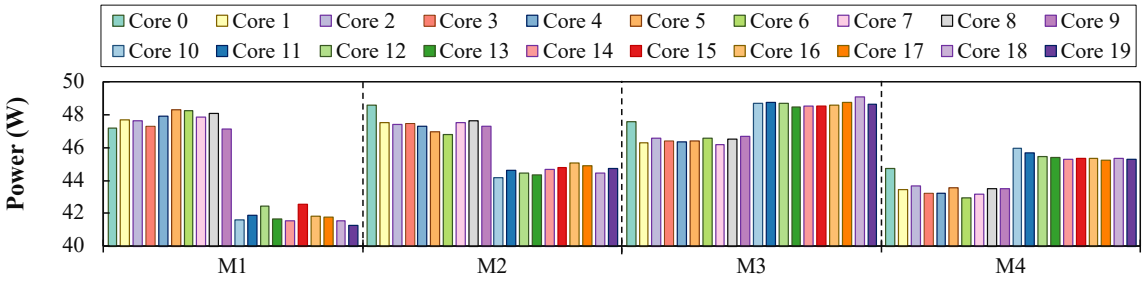

Figura 1. Potência de Pico de cada núcleo em quatro máquinas idênticas ao executar a mesma tarefa

frequência de *soft-errors*, que podem levar a falhas no sistema, enquanto também reduzem sua expectativa de vida [Marques et al. 2023]. Portanto, reduzir a potência dissipada por componente torna-se de suma importância, pois isso não só impacta a economia dos servidores de nuvem, mas também o meio ambiente ao mesmo passo que reduz comportamentos indesejados do sistema e prolongando a longevidade dos componentes de *hardware* [Navaux et al. 2023].

Nesse cenário, RLP oferece uma opção promissora para reduzir a potência dissipada em servidores na nuvem. Utilizando a capacidade de processamento paralelo de múltiplos recursos (por exemplo, unidades de processamento e de armazenamento) dentro da infraestrutura do servidor, a potência pode ser reduzida significativamente sem impactar negativamente o desempenho. Essa abordagem também possibilita o emprego de *dynamic power scaling*, onde recursos inativos ou pouco utilizados podem ser seletivamente desativados ou colocados em estados de baixo consumo, conservando potência. Além disso, o RLP facilita o balanceamento de recursos, reduzindo a necessidade de provisionamento excessivo de *hardware* e assim minimizando requisitos de potência. No fim, ao empregar RLP, servidores na nuvem podem atingir níveis ótimos de eficiência energética, garantindo uma operação sustentável enquanto satisfazem as crescentes demandas dos serviços da nuvem.

No entanto, o comportamento das unidades de processamento não é uniforme devido a variabilidade de processos *within-die* observada em tecnologias modernas, o que acaba complicando as alocações de tarefas nos núcleos. A presença de variabilidade pode causar diferenças significativas em cada núcleo em termos de dissipação de potência e a frequência máxima que eles podem sustentar. Em processadores comerciais, a frequência de operação máxima de todos os núcleos (conhecida como frequência nominal) é restringida e determinada pelo núcleo mais lento. Então, por mais que não seja possível observar diretamente a discrepância na frequência de operação proveniente da variabilidade, ela tem um impacto direto na potência dissipada, como ilustrado na Figura 1. A Figura mostra a potência máxima de cada núcleo em quatro arquiteturas *2x 10-core Intel Xeon* (*M1*, *M2*, *M3*, e *M4*, como descritos na Seção 4) onde cada núcleo executa exatamente a mesma tarefa dentro das mesmas condições. Conforme pode-se observar, existe uma variabilidade significativa na potência entre os processadores e mesmo nos núcleos de um mesmo processador em específico - principalmente quando compara-se diferentes nodos NUMA (*Non-Uniform Memory Accesses*, representados pelos núcleos *0* a *9* e núcleos *10* a *19*).

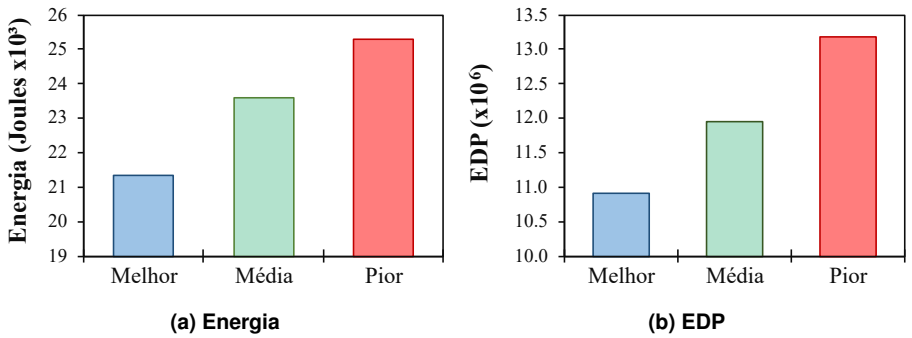

Figura 2. Energia e EDP do pior, médio, e melhor caso dos núcleos da Figura 1.

Essa diferença na potência afeta o consumo de energia de todo o sistema e o custo-benefício entre desempenho e energia, representado pela métrica EDP (*energy-delay product*), como demonstrado na Figura 2. Ela mostra o consumo de energia e EDP dos núcleos com os maiores e menores consumos de energia (melhores e piores casos) e a média através de todos os núcleos da Figura 1 em um ambiente despreocupado com a variabilidade. Como observado, a diferença no consumo de energia entre o pior e o melhor caso para executar a mesma tarefa é de aproximadamente 16% enquanto a diferença de EDP é de aproximadamente 19%. Tendo em mente a discussão anterior, nós propomos *PowerSaver*, uma estratégia que considera os efeitos da variabilidade de processo na potência de cada núcleo para otimizar a eficiência energética de servidores na nuvem. Ela minimiza a potência dissipada por um servidor na nuvem sem aumentar o tempo de execução das tarefas computacionais através do mapeamento inteligente de tarefas para os núcleos e gerenciamento eficiente da frequência/tensão de cada núcleo e do sistema de memória.

PowerSaver é dividido em dois módulos principais: *profiling da arquitetura* e *adaptação dinâmica*. O primeiro gera um mapa de variabilidade de potência de cada núcleo, que é armazenado em um banco de dados para ser usado pela adaptação dinâmica. Com essa informação, os núcleos são classificados de acordo com suas características de variabilidade. Assim, a *adaptação dinâmica* coleta métricas de *hardware* de novas tarefas na fila de execução e que não foram classificadas ainda, para mapeá-las para o núcleo mais apropriado. A frequência de *core/uncore* também é automaticamente ajustada para reduzir a dissipação de potência sem impacto no desempenho da tarefa. Assim, *PowerSaver* implementa um banco de dados para armazenar a classificação da arquitetura e as características de cada tarefa já executada. Com isso, a *adaptação dinâmica* é acelerada pois as tarefas que já foram classificadas não precisam ser reclassificadas.

Nós validamos *PowerSaver* executando uma lista de vinte tarefas de domínios diferentes em quatro processadores Intel *multicore* em cenários diferentes de acordo com o nível de RLP: *25% RLP*, *50% RLP*, *75% RLP*, e *100% RLP*, onde cada um representa a quantidade de recursos de hardware usados em paralelo. Comparado à execução padrão de tarefas nos servidores de nuvem, *PowerSaver* reduz a potência de pico em até 25.6% com mínimo impacto no desempenho das tarefas. Nós também demonstramos que *PowerSaver* apresenta melhores resultados que *HiMap* [Rathore et al. 2018], uma estratégia conhecida do estado da arte.

2. Trabalhos Relacionados

Trabalhos que buscam otimizar a eficiência energética através da redução de potência dos componentes de hardware são discutidos a seguir. R. Teodorescu e J. Torrellas [Teodorescu and Torrellas 2008] propõem algoritmos de escalonamento que consideram a variabilidade de processo para maximizar o desempenho a um limite de potência. J. Winter e D. Albonesi [Winter and Albonesi 2008] propõem algoritmos de escalonamento para melhorar o desempenho e consumo de energia de núcleos que apresentam frequências heterogêneas. S. Garg et al. [Garg et al. 2010] discutem estratégias para mitigar o impacto de variabilidade de processo no desempenho e consumo de energia em plataformas *multicore*.

S. Dighe et al. [Dighe et al. 2011] propõem um esquema de mapeamento de *threads* e DVFS para melhorar a eficiência energética em processadores *multicore*. B. Raghunathan et al. [Raghunathan et al. 2013] propõem um algoritmo para seleção ótima de núcleos e mapeamento de *threads* para melhorar o desempenho de aplicações paralelas. D. Stamoulis e D. Marculescu [Stamoulis and Marculescu 2016] apresentam uma estratégia para mapeamento de *threads* em arquiteturas heterogêneas que satisfaça necessidades de desempenho por aplicação em cenários onde há variabilidade de processo e carga de trabalho. HiMap [Rathore et al. 2018] é uma abordagem dinâmica que ajusta componentes de hardware para satisfazer demandas de desempenho, potência e térmicas. da Silva et al,. [da Silva et al. 2023] analisa a execução de múltiplas aplicações de maneira concorrente, porém, não explora a variabilidade de processo.

Embora diferentes estratégias têm sido propostas, elas não aplicam mapeamento de tarefas e ajuste dinâmico de frequência/tensão dos subsistemas de *core*/*uncore* ao mesmo tempo para otimizar a potência com mínimo impacto no desempenho das tarefas. Assim, *PowerSaver* considera as características intrínsecas de cada tarefa durante a execução para encontrar o núcleo mais adequado para executar a tarefa e explora os domínios de frequência do *core* e *uncore* para reduzir ainda mais a potência sem impactar negativamente no desempenho.

3. *PowerSaver*

PowerSaver objetiva reduzir a potência dissipada em servidores de alto desempenho na nuvem sem penalizar o desempenho das tarefas. Para isto, ela explora a variabilidade intrínseca de potência entre os núcleos computacionais em cada nodo NUMA para aplicar mapeamento de tarefas e ajuste da frequência do *core* e *uncore*. O fluxo de execução de *PowerSaver* é ilustrado na Figura 3 e discutido a seguir.

3.1. Fluxo de Execução

3.1.1. Inicialização

A infraestrutura de nuvem disponibiliza um conjunto específico de tarefas que precisam ser executadas, independentemente de serem solicitadas pelo mesmo usuário ou por usuários diferentes, ou seja, é um ambiente *multitenant*. Essas tarefas alimentam *PowerSaver* juntamente com seus respectivos conjuntos de dados a serem processados. Nesta etapa, *PowerSaver* também verifica a disponibilidade e a atualização do mapa de variabilidade de potência, que é gerado pelo módulo de *criação estática de perfis da arquitetura*

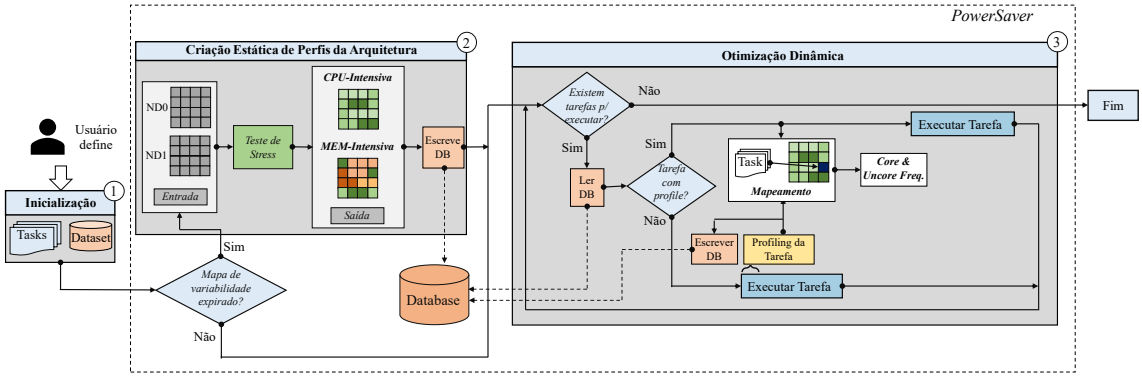

Figura 3. Fluxo de Execução do *PowerSaver*

na Etapa 2. Esse mapa contém informações sobre a estimativa de potência e frequência máxima de cada núcleo da arquitetura de destino. Caso exista necessidade de atualização, o módulo é acionado para criar um novo mapa de variabilidade de cada núcleo. Caso contrário, *PowerSaver* avança para a Etapa 3 (otimização dinâmica) sem realizar o perfil da arquitetura. Vale destacar que o mapa de variabilidade só precisa ser gerado a cada três-seis meses, uma vez que o processo de envelhecimento e alteração do comportamento dos núcleos é gradual e lento [Gnad et al. 2015].

3.1.2. Criação de Perfis da Arquitetura

Nesta etapa, *PowerSaver* submete cada núcleo a um teste de estresse a fim de obter informações sobre seu perfil de dissipação de potência, seguindo o Algoritmo 1 e observando a Figura 3. A arquitetura alvo é composta por tnd nodos NUMA representados por $ND = \{ND_0, \ldots, ND_{tnd-1}\}$, e cada nó NUMA contém nc núcleos representados por $C = \{C_0, \ldots, C_{nc-1}\}$. O algoritmo começa inicializando as estruturas de dados necessárias para a execução: o mapa de variabilidade $varMap$, que armazena informações sobre a potência de pico e frequência de operação máxima para cada núcleo; e $idlePower$, que armazena a potência em que cada núcleo está em *idle*, garantindo que todos os testes de estresse iniciem sob as mesmas condições. Em seguida, a potência em *idle* é medida para cada núcleo c em cada nó NUMA nd. Vale ressaltar que a função $sleep$ dura em média 10s.

Uma vez que a potência atual de cada núcleo atinge o valor de $idlePower$, o algoritmo inicia o teste de estresse (*linha 13*). Esse teste consiste na execução de uma tarefa sintética composta por duas fases que submetem tanto o sistema de memória (por meio da aplicação *stream*) quanto a CPU (por meio do cálculo do número Pi) a um estresse controlado, até que a potência de pico se estabilize. Ao final do teste de estresse para cada núcleo, o mapa de variabilidade é atualizado com as informações sobre a potência de pico de cada núcleo, e esses dados são armazenados na respectiva posição da estrutura de dados $varMap$. Posteriormente, todos os núcleos são classificados em duas categorias (*mais saudáveis* e *mais vulneráveis*) e ordenados de acordo com suas potências de pico. Essas informações serão utilizadas pela fase de otimização dinâmica (*Etapa 3*) para determinar qual núcleo é o mais adequado para executar a tarefa em questão.

Algorithm 1 Criação Estática de Perfis da Arquitetura

Input: $ND \leftarrow \{ND_0, ND_1, \ldots, ND_{tnd-1}\}$: Conjunto de nodos NUMA
$C \leftarrow \{C_0, C_1, \ldots, C_{nc-1}\}$: Conjunto de núcleos por nodo NUMA

1: $varMap[tnd][nc] \leftarrow \varnothing$
2: $idlePower[tnd][nc] \leftarrow \varnothing$
3: **for** *cada nodoNUMA* ***nd*** *in* ***tnd*** **do**
4: **for** *cada núcleo* ***c*** *in* ***nc*** **do**
5: $idlePower[nd][c] \leftarrow$ *getIdlePower(nd, c)*
6: **end for**
7: **end for**
8: **for** *cada nodoNUMA* ***nd*** *in* ***tnd*** **do**
9: **for** cada núcleo c in C **do**
10: **while** $getCurrPower(nd, c) > idlePower[nd][c]$ **do**
11: *sleep*(μ)
12: **end while**
13: $varMap[nd][c] \leftarrow$ *stressTest(nd, c)*
14: **end for**
15: **end for**
16: *Ordena_Classifica*(*Map*)
17: **return** $varMap$

3.1.3. Otimização Dinâmica

Nesta etapa, *PowerSaver* realiza um mapeamento dinâmico *task-to-core* e ajusta a frequência do *core* e *uncore* com base nas características da tarefa (CPU- ou Memória-Intensiva), conforme ilustrado na Figura 3(3). Assim, quando há tarefas para executar no servidor da nuvem, *PowerSaver* inicia lendo a base de dados para obter o mapa de variabilidade previamente coletado. Com essas informações disponíveis, *PowerSaver* determina o fluxo de execução da tarefa.

Se a tarefa é executada pela primeira vez na arquitetura, *PowerSaver* inicia um perfil dinâmico específico para a tarefa (*criação de tarefas* na Figura 3(3)). O perfil é criado através das seguintes etapas: (*a*) A tarefa é executada usando o escalonador padrão do sistema operacional Linux (CFS - *Completely Fair Scheduler*), e as frequências de núcleo/não núcleo são configuradas para ajustar de acordo com a carga de trabalho da CPU/memória. (*b*) Durante a execução, *PowerSaver* coleta informações sobre o número de instruções por ciclo (IPC) e o uso de memória, especialmente o comportamento da memória *cache* L2 e L3. Esses dados são medidos até que as variações não ultrapassem 20% entre as medições realizadas a cada segundo. O comportamento da memória cache e IPC são utilizados para classificar tarefas com uso intensivo da CPU ou de memória, onde um IPC maior indica uma tarefa mais intensiva em CPU, enquanto um maior número de acessos ao cache L3 junto com um IPC baixo indica uma tarefa mais intensiva em memória. (*c*) Após a coleta das informações, *PowerSaver* interrompe o *profiling* da tarefa e armazena os dados (IPC e comportamento das memórias cache L2/L3) no banco de dados para serem utilizados na próxima execução da mesma tarefa, evitando o processo de *profiling* novamente. Em seguida, *PowerSaver* realiza o mapeamento tarefa-a-núcleo e gerencia as frequências dos núcleos/uncore com base nas características da tarefa e na classificação dos núcleos feita no *Estágio 2*, como descrito a seguir.

PowerSaver considera que os nós NUMA com menor potência de pico são mais

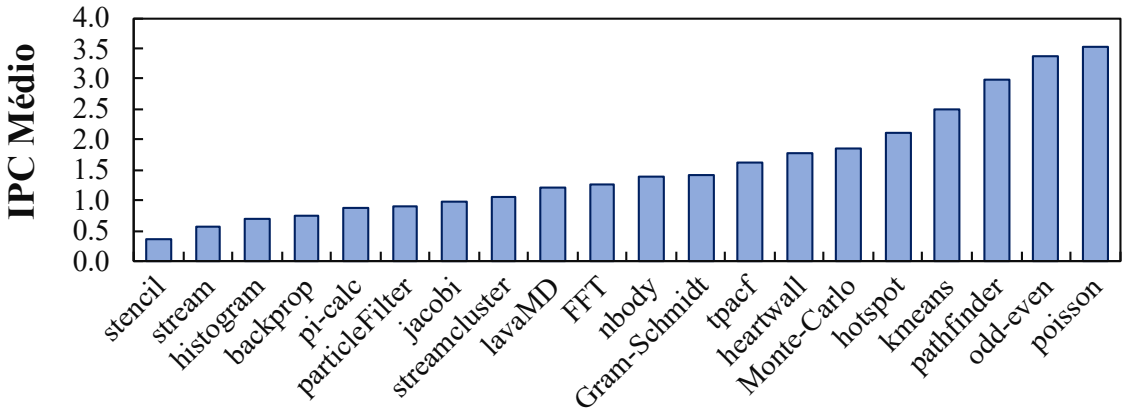

Figura 4. IPC médio de cada tarefa utilizada para validar *PowerSaver*

adequados para executar tarefas *CPU-Intensivas*, pois possuem mais margem para ajustar a frequência do núcleo ao máximo permitido dentro dos limites de TDP (*thermal design power*). Além disso, tarefas *CPU-Intensivas* requerem menos operações de memória, o que permite definir a frequência do sistema *uncore* para o mínimo, reduzindo a potência total sem afetar o desempenho da tarefa. Por outro lado, os nós NUMA com potência de pico mais alta são mais adequados para executar tarefas de uso intensivo de memória, pois a CPU estará menos ativa. Nesse caso, a frequência do núcleo pode ser ajustada de acordo com a carga de trabalho, reduzindo a potência dissipada e permitindo aumentar a frequência do núcleo para acelerar as operações de memória sem aumentar a potência de pico do núcleo.

4. Metodologia da Análise Experimental

Tarefas/Aplicações. Nós consideramos 20 tarefas/aplicações já implementadas em C/C++ de diferentes suítes. **Três do Parboil [Stratton et al. 2012]**: *HISTO*, *STENCIL* e *TPACF*. **Oito do Rodinia [Che et al. 2009]**: *backprop*, *heartwall*, *hotspot*, *kmeans*, *lavaMD*, *particle filter*, *pathfinder* e *streamcluster*. **Nove de diferentes domínios:** *fast Fourier transform* (FFT), *Gram-Schnmidt*, *Jacobi*, *Monte-Carlo*, *N-body*, *Ordenação odd-even*, *cálculo do pi*, *Equação de Poisson*, *Stream*. Cada tarefa foi executada com o conjunto de entrada padrão definido em cada suíte. Adicionalmente, conforme mostrado na Figura 4, elas representam diferentes comportamentos com relação ao número médio de instruções por ciclo (IPC), refletindo o quão intensiva cada tarefa é em termos de CPU ou memória.

Ambiente de Execução. Os experimentos foram realizados em quatro arquiteturas *multicore* idênticas (*M1*, *M2*, *M3* e *M4*) com as seguintes configurações: *2x 10-core Intel Xeon E5-2650 v3*, cada núcleo operando em frequência de $1.2GHz$ até $2.3GHz$ ($3.0GHz$ quando o *Turbo Boost* está ativo). Cada núcleo possuí uma cache L1 de 32KB e L2 de 256KB, e compartilha 2x25MB de cache L3 e 2x64GB de RAM. Adicionalmente, o frequência do *uncore* pode variar de $1.2GHz$ até $3.0GHz$. Todas as arquiteturas estavam com o *Linux Kernel v.4.19.0-21* instalado. Foram utilizados *hardware counters* através do *perf tool* do *Linux* para medir o consumo de energia. Por fim, cada tarefa foi compilada com GCC/G++ 10.2 usando a *flag* de otimização *-O3*.

Conjunto de Experimentos. Nós comparamos *PowerSaver* com as seguintes estratégias: ***CFS***, onde as tarefas são alocadas pelo sistema operacional *Linux* usando o *Completely Fair Scheduler*, que compartilha o tempo de execução com todas as tarefas e visa o equilíbrio de uso de recursos; ***HiMap***, uma conhecida abordagem de mapeamento que visa satisfazer restrições de potência, desempenho e térmicas [Rathore et al. 2018]. Nós implementamos o mecanismo *HiMap* em linguagem C e aplicamos em nossos experimentos. Considerando que a carga de trabalho dos servidores em nuvem pode variar de acordo com vários fatores (por exemplo, número de usuários, tipo de tarefa e contrato de nível de serviço), avaliamos todas as estratégias em quatro cenários diferentes, de acordo com o grau de RLP explorado pela arquitetura: **25%-RLP** - o tamanho do *batch* corresponde a um quarto do número total de núcleos da arquitetura. As seguintes tarefas foram consideradas: *HISTO*, *STENCIL*, *TPACF*, *ordenação odd-even* e *Equação de Poisson*. Além disso, esse é o cenário que menos estressa a arquitetura e oferece mais espaço para otimização. **50%-RLP** - o tamanho do *batch* é igual à metade do número de núcleos disponíveis. Consideramos as seguintes tarefas: *STENCIL*, *Stream*, *Backprop*, *Jacobi method*, *Streamcluster*, *N-body*, *Gram-Schmidt*, *Monte-Carlo*, *Hotspot* e *ordenação odd-even*. **75%-RLP** - o tamanho do *batch* corresponde a três quartos do número de núcleos. As seguintes tarefas foram executadas: *STENCIL*, *HISTO*, *Stream*, *Backprop*, *Cálculo do Pi*, *Filtro de partículas*, *método Jacobi*, *Heartwall*, *Monte-Carlo*, *Hotspot*, *lavaMD*, *Kmeans*, *Path Finder*, *Ordenação odd-even* e *Equação de Poisson*. **100%-RLP** - o tamanho do *batch* considera todas as tarefas. Cabe ressaltar que esse cenário é o que mais estressa a arquitetura e oferece o menor espaço para otimização.

5. Resultados

A Figura 5 ilustra a distribuição dos valores de potência (Watts) para todos os núcleos em cada arquitetura multicore (*M1*, *M2*, *M3* e *M4*) e cenário (*25%-RLP*, *50%-RLP*, *75%-RLP* e *100%-RLP*). Para cada *box*, os bigodes na parte inferior e superior representam os valores mínimo e máximo, respectivamente. O interior de cada *box* é dividido em três partes: o primeiro quartil (*Q1*), a mediana (*Q2*) e o terceiro quartil (*Q3*). Finalmente, o intervalo entre dois quartis diferentes representa *25%* da distribuição.

Começamos destacando que a política de escalonamento adotada pelo Sistema Operacional Linux instalado em servidores em nuvem (*CFS*) não é capaz de fornecer resultados de potência competitivos na maioria das vezes em comparação com *HiMap* e *PowerSaver*. A principal razão para este comportamento é que o *CFS* não considera a variabilidade de potência entre os núcleos ao mapear a tarefa para execução. Em vez disso, ele compartilha o tempo de execução para todas as tarefas visando um equilíbrio de uso de recursos. Além disso, a potência dissipada total pelo *package* também é afetada pela forma como a frequência do núcleo e do *uncore* é configurada na execução padrão das tarefas. Como eles geralmente são configurados para atingir a frequência máxima possível para garantir o acordo de nível de serviço da nuvem (por exemplo, capacidade de resposta a sistemas e aplicações), a dissipação de potência é maior que o *HiMap* e o *PowerSaver*. Ilustramos esse comportamento na Figura 6 quando cada estratégia executa o cenário com *25%* de exploração de RLP no sistema *M4*.

Ao considerar os resultados obtidos por *HiMap*, ele supera o *CFS* na maioria dos casos, pois prioriza o mapeamento de tarefas para os núcleos saudáveis primeiro e aplica DVFS no domínio do núcleo para otimizar ainda mais a potência. No entanto, o fato de

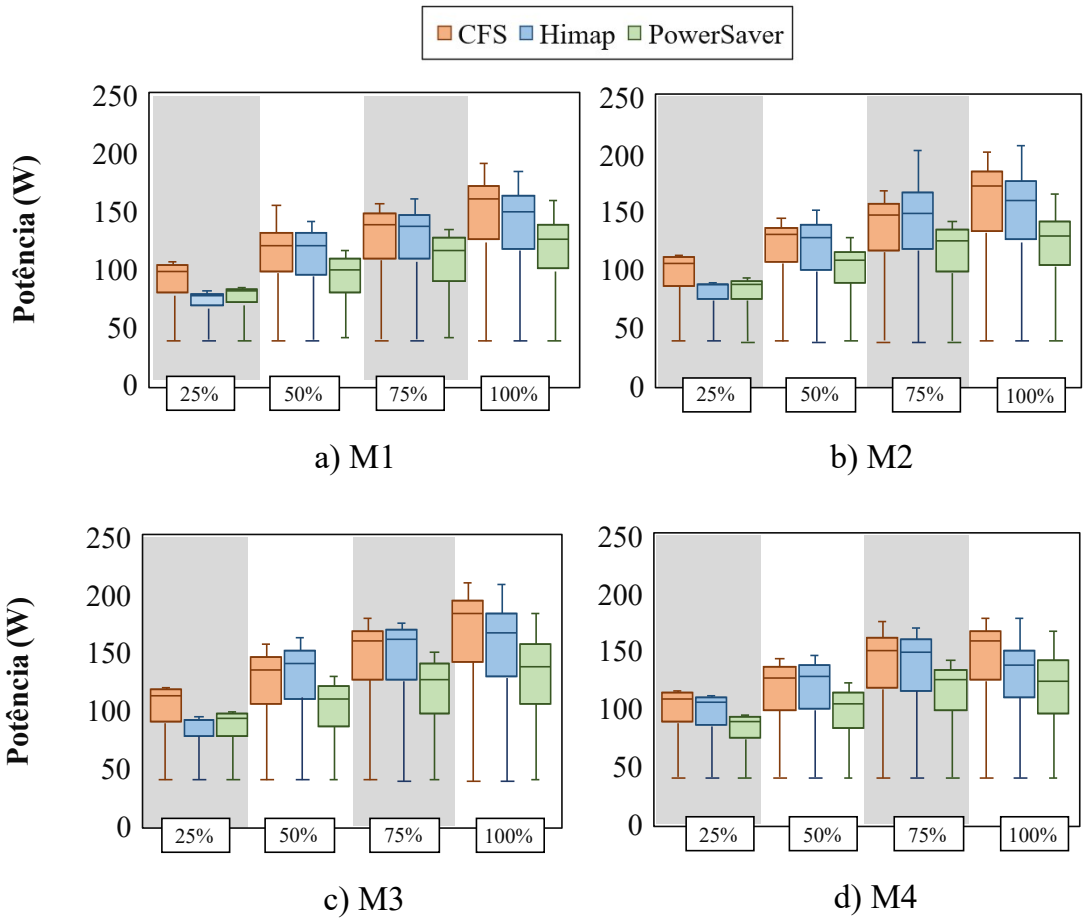

Figura 5. Distribuição dos resultados de potência para todas as estratégias executando cada cenário (↓ valores = melhores resultados)

não considerar o gerenciamento de frequência do subsistema *uncore* destaca a necessidade de uma solução mais abrangente. Assim, *PowerSaver* preenche esta lacuna mapeando sinergicamente a tarefa para o núcleo mais adequado para executá-la e aplicando DVFS e UFS de acordo com as características intrínsecas da tarefa. Neste cenário, as reduções na dissipação de potência (Figura 6), se deve às seguintes decisões, conforme já discutido: (*i*) quando é uma tarefa intensiva de memória, *PowerSaver* diminui a frequência/tensão do núcleo para ter uma redução correspondente na potência; e (*ii*) no caso de uma tarefa CPU intensiva, *PowerSaver* define a frequência do *uncore* para o mínimo permitido para reduzir ainda mais a potência dissipada. Em ambas as decisões, o impacto no desempenho é mínimo (como será discutido nas próximas seções), pois as modificações na frequência não afetam as principais características da tarefa. Com isso, *PowerSaver* reduz a potência de pico em até *25.6%* em relação ao CFS (*M2*, 25%-RLP) e *20.3%* em relação ao HiMap (*M2*, *75%-RLP*).

Um desafio significativo ao propor técnicas de otimização de eficiência energética para sistemas de alto desempenho na nuvem é evitar a degradação do desempenho das tarefas. Assim, para demonstrar que a redução na potência dissipada não interfere no desempenho e consumo de energia das tarefas, nós coletamos também o tempo de execução dos cenários apresentados. Os resultados de tempo (em segundos) e consumo de energia

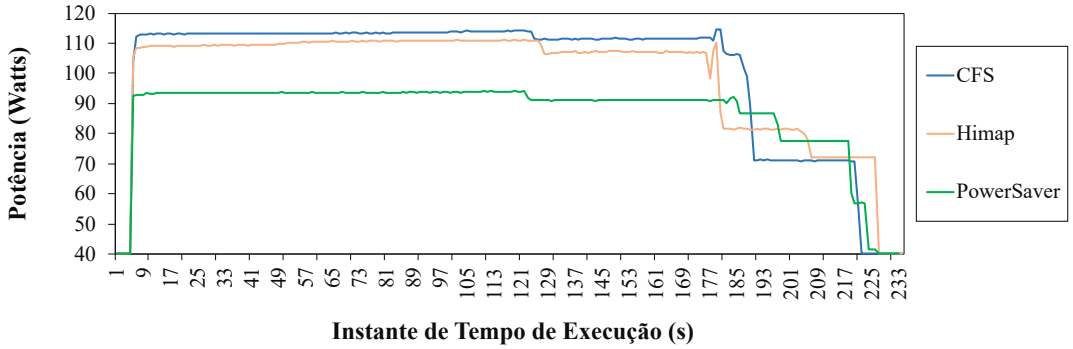

Figura 6. Potência dissipada por segundo quando cada estratégia executa o cenário 25%-RLP na máquina M4.

(em Joules) para executar cada cenário por completo em cada máquina são mostrados nas Tabelas 1 e 2. Conforme observado, uma vez que *PowerSaver* realiza o mapeamento da tarefa para cada núcleo considerando as características da tarefa e do núcleo alvo ao mesmo tempo que ajusta a frequência/tensão dos núcleos e *uncore* de modo que apenas a potência seja reduzida, o impacto no desempenho das tarefas é de apenas *2.9%* comparado a política *CFS* e *1.3%* comparado ao *HiMap*.

Ao ser capaz de manter um desempenho similar à estratégias que utilizam todo o poder computacional disponível das arquiteturas sem se preocupar com a potência dissipada, *PowerSaver* também apresenta melhores resultados de consumo de energia (Tabela 2). No caso mais significativo, consumo de energia é reduzido em *17%* comparado ao *CFS* (*75%-RLP* na *M3*) e *16%* ao *HiMap* (*50%-RLP* na *M2*). Na média de todos os cenários e máquinas, o consumo de energia foi reduzido em *11%* e *8%* comparado a *CFS* e *HiMap*, respectivamente.

5.1. Análise de *Overhead*

O *overhead* de *PowerSaver* vem de três situações. (*i*) toda vez que um novo perfil estático da arquitetura precisa ser gerado, o teste de estresse para obter o mapa de variabilidade de potência de cada núcleo é realizado. Esse teste leva cerca de 3 minutos por núcleo, o que descobrimos experimentalmente ser o período que o núcleo precisa para atingir sua potência máxima ao executar uma tarefa serial. Portanto, o *overhead* é de apenas $3min\times$ *número de núcleos*, o que é insignificante em comparação aos benefícios de empregar *PowerSaver*. Mesmo que o número de núcleos disponíveis na arquitetura de destino seja grande, esse procedimento só precisa ser feito uma vez a cada vários meses (por exemplo, 3 ou 6 meses), pois a variação de potência de cada núcleo decorrente do envelhecimento é relativamente lento. (*ii*) quando uma nova tarefa inicia a execução, o perfil dela deve ser extraído e as características armazenadas no banco de dados. Como essa etapa é executada à medida que a tarefa é executada, o *overhead* é compensado. (*iii*) além disso, há também um *overhead* implícito com relação ao banco de dados e ao tempo para acessá-lo. Enquanto *PowerSaver* ocupa *8,4 Kb* de espaço e cada *hash* com as características da tarefa armazenada adiciona *141 bytes*, o tempo para atualizar o banco de dados é de apenas *0,00098s* e o tempo para pesquisar a configuração de uma determinada tarefa é *0.00182s*.

Tabela 1. Tempo total de execução de cada estratégia em cada cenário. Valores são mostrados em segundos.

	25%-RLP			50%-RLP			75%-RLP			100%-RLP		
	CFS	*Himap*	*PowerSaver*	*CFS*	*Himap*	*PowerSaver*	*CFS*	*Himap*	*PowerSaver*	*CFS*	*Himap*	*PowerSaver*
M1	229	233	236	266	272	273	273	281	283	365	370	374
M2	229	234	236	266	272	275	279	285	289	366	371	373
M3	228	227	236	266	273	279	278	281	286	364	370	371
M4	229	227	236	287	290	292	278	281	287	372	379	379

Tabela 2. Consumo de energia de cada estratégia em cada cenário. Valores são mostrados em *Joules*

	25%-RLP			50%-RLP			75%-RLP			100%-RLP		
	CFS	*Himap*	*PowerSaver*	*CFS*	*Himap*	*PowerSaver*	*CFS*	*Himap*	*PowerSaver*	*CFS*	*Himap*	*PowerSaver*
M1	20621.2	19203.8	17738.5	31009.6	30777.8	27032.4	36517.5	36752.8	33026.5	49763.9	47425.9	45485.3
M2	22320.0	20164.4	19947.6	34113.0	32691.1	32960.5	39357.1	38272.3	37176.6	51401.5	51370.5	47400.9
M3	23691.2	21453.1	20555.8	33904.9	32917.3	31761.9	43016.3	41635.5	35979.3	52957.9	52561.0	48274.9
M4	23116.8	22906.6	20390.7	34046.9	34590.7	29319.9	40102.2	38613.2	36772.1	50222.1	50397.4	43090.8

6. Conclusão e Trabalhos Futuros

Neste artigo, nós apresentamos *PowerSaver*, uma estratégia que aplica sinergicamente o mapeamento de tarefas para cada núcleo e gerencia os níveis de frequência e tensão dos domínios de *core* e *uncore* para otimizar a eficiência energética dos servidores em nuvem. Ao executar diferentes cenários em quatro arquiteturas multicore idênticas, mostramos que *PowerSaver* pode reduzir a potência de pico em até 25.6% e 20.3% de servidores em nuvem em comparação com a execução padrão de tarefas e uma abordagem conhecida do estado da arte. Adicionalmente, mostramos que este benefício apresenta um mínimo impacto no desempenho das tarefas, contribuindo para reduzir também o consumo de energia do sistema e melhorar o custo-benefício entre desempenho e energia. Em trabalhos futuros, pretendemos aprimorar *PowerSaver* para explorar a variabilidade de potência entre todas as máquinas em um *warehouse* ao executar o mapeamento de tarefa para núcleo.

Referências

Bohr, M. (2007). A 30 year retrospective on dennard's mosfet scaling paper. *IEEE Solid-State Circuits Society Newsletter*, 12(1):11–13.

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 ISWC*, pages 44–54. 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. (2023). Smart resource allocation of concurrent execution of parallel applications. *Concurrency and Computation: Practice and Experience*, 35(17):e6600.

Dighe, S., Vangal, S. R., Aseron, P., Kumar, S., Jacob, T., Bowman, K. A., Howard, J., Tschanz, J., Erraguntla, V., Borkar, N., De, V. K., and Borkar, S. (2011). Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor. *IEEE Journal of Solid-State Circuits*, 46(1):184–193.

Garg, S., Marculescu, D., and Herbert, S. X. (2010). Process variation aware performance modeling and dynamic power management for multi-core systems. In *2010 IEEE/ACM Int. Conf. on Computer-Aided Design (ICCAD)*, pages 89–92.

Gnad, D., Shafique, M., Kriebel, F., Rehman, S., Sun, D., and Henkel, J. (2015). Hayat: Harnessing dark silicon and variability for aging deceleration and balancing. In *52nd ACM/EDAC/IEEE DAC*, pages 1–6. IEEE.

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.

Marques, S. M. V. N., Rossi, F. D., Luizelli, M. C., Beck, A. C. S., and Lorenzon, A. F. (2023). Seamless thermal optimization of parallel workloads. *IEEE Design Test*, 40(5):34–41.

Navaux, P. O. A., Lorenzon, A. F., and da Silva Serpa, M. (2023). Challenges in high-performance computing. *Journal of the Brazilian Computer Society*, 29(1):51–62.

Raghunathan, B., Turakhia, Y., Garg, S., and Marculescu, D. (2013). Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors. In *DATE*, pages 39–44.

Rathore, V., Chaturvedi, V., Singh, A. K., Srikanthan, T., Rohith, R., Lam, S.-K., and Shaflque, M. (2018). Himap: A hierarchical mapping approach for enhancing lifetime reliability of dark silicon manycore systems. In *DATE*, pages 991–996.

Stamoulis, D. and Marculescu, D. (2016). Can we guarantee performance requirements under workload and process variations? In *ISLPED*, page 308–313, New York, NY, USA. ACM.

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.

Teodorescu, R. and Torrellas, J. (2008). Variation-aware application scheduling and power management for chip multiprocessors. In *2008 Int. Symposium on Computer Architecture*, pages 363–374.

Winter, J. A. and Albonesi, D. H. (2008). Scheduling algorithms for unpredictably heterogeneous cmp architectures. In *2008 IEEE Int. Conf. on Dependable Systems and Networks With FTCS and DCC (DSN)*, pages 42–51.