Gerenciamento de Recursos para Execução Energeticamente Eficiente de *Workloads* em *Data Centers* com CPUs Híbridas

Marcelo K. Moori[1], Felipe N. Laguna[1], Hiago Mayk G. de A. Rocha[2], Antonio Carlos S. Beck[1,3]

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

[2]Laboratório Nacional de Computação Científica (LNCC), Petrópolis - RJ, Brasil

[3]Center for Embedded Devices and Research in Digital Agriculture (CEDRA), SENAI-RS, Brasil

[1]{mkmoori, fnlaguna, caco}@inf.ufrgs.br, [2]mayk@lncc.br

Abstract. *Asymmetric multicore processors (AMPs) combine high-performance and energy-efficient cores under a shared instruction set architecture and are increasingly being adopted in datacenters for compute-intensive workloads. However, AMPs challenge conventional container resource management mechanisms, which typically assume homogeneous cores. In this paper, we propose* COREWISE*, an adaptive resource manager for the energy-efficient allocation of containerized workloads on AMP systems. Based on offline profiling,* COREWISE *adapts each application's thread-level parallelism by reducing the number of threads when appropriate, ranks feasible core allocations according to the Energy-Delay Product (EDP), and applies a cooperative three-phase strategy: (i) priority-driven best-fit allocation, (ii) cooperative resource reduction, and (iii) residual core redistribution. We evaluate* COREWISE *using NAS Parallel Benchmarks running in Docker on an Intel Core i9-12900K (Alder Lake) under a highly contended scenario with seven candidate configurations per workload and 50 concurrent containers. Compared with the serial baseline,* COREWISE *reduced the average EDP by 91.2%, outperforming classical container scheduling heuristics by 56.51% and a homogeneous variant that ignores core heterogeneity by 49.92%.*

Resumo. *Processadores multicore assimétricos (AMPs) combinam cores de alto desempenho e de eficiência energética sob uma arquitetura de conjunto de instruções compartilhada e vêm sendo adotados em datacenters para cargas de trabalho cada vez mais intensivas. Entretanto, AMPs desafiam os mecanismos convencionais de gerenciamento de containers, que normalmente assumem cores homogêneos. Neste artigo, propomos o* COREWISE*, um gerenciador adaptativo para alocação energeticamente eficiente de workloads conteinerizados em sistemas AMPs. Com base em profiling, o* COREWISE *adapta o paralelismo em nível de thread de cada aplicação (reduzindo o número de threads quando necessário), classifica alocações viáveis de cores pelo Energy-Delay Product (EDP) e aplica uma estratégia cooperativa em três fases: (i) alocação best-fit orientada por prioridade, (ii) redução cooperativa e (iii) redistribuição de cores residuais. Avaliamos o* COREWISE *com kernels do NAS Parallel Benchmark Suite executados em Docker sobre um Intel Core i9-12900K (Alder Lake), em um cenário de alta contenção com 7 configurações candidatas por workload*

e 50 containers concorrentes. Em comparação com o baseline serial, o CoreWise *reduziu o EDP em 91,2% em média, superando heurísticas clássicas para containers em 56,51% e uma variante que desconsidera a heterogeneidade em 49,92%.*

1. Introdução

A crescente adoção de processadores multicore assimétricos (AMPs) está remodelando a computação moderna. Arquiteturas no estilo big.LITTLE, que combinam cores de alto desempenho e eficiência energética (P-cores e E-cores, respectivamente, na terminologia usada pela Intel) [Bilbao et al. 2023], incluindo Intel Alder Lake/Meteor Lake e Apple M-series (M1-M4), vêm sendo amplamente empregadas em ambientes cliente, *edge* e nuvem [Schöne et al. 2024a, Laguna et al. 2026]. Seu objetivo é equilibrar desempenho e eficiência energética por meio do mapeamento adequado de *workloads* para cores heterogêneos.

Paralelamente, *containers* (por exemplo, Docker, Podman e Apptainer) tornaram-se o modelo *de facto* para implantação de aplicações em nuvem e, cada vez mais, em *datacenters* e Computação de Alto Desempenho (HPC), oferecendo isolamento leve, inicialização rápida e desempenho próximo ao nativo [Liu and Guitart 2022]. Entretanto, *runtimes* de *containers* e escalonadores do sistema operacional geralmente assumem cores homogêneos, ignorando a assimetria das arquiteturas AMP [Kwon et al. 2011a]. Como consequência, *workloads* paralelos podem apresentar comportamentos significativamente distintos em P-cores e E-cores, tornando estratégias tradicionais de escalonamento e alocação de *threads* ineficientes em termos de desempenho e consumo de energia.

O problema se intensifica quando múltiplos *containers* compartilham o mesmo processador, competindo pelos poucos P-cores enquanto E-cores permanecem subutilizados. Embora mecanismos de *autoscaling* ajustem recursos com base em métricas agregadas, como utilização de CPU e memória [Balla et al. 2020], eles não consideram conjuntamente o paralelismo em nível de *thread* (TLP), a heterogeneidade dos cores e a coexecução de *workloads*. Além disso, embora provedores como a AWS disponibilizem instâncias baseadas em CPUs híbridas (por exemplo, `mac2-m1ultra.metal` e `mac2-m2pro.metal`) [aws 2024], essas soluções concentram-se no provisionamento de *hosts* dedicados e não exploram o potencial de otimização intranó proporcionado por políticas de escalonamento conscientes da heterogeneidade.

Neste trabalho, propomos o CoreWise, um gerenciador para alocação energeticamente eficiente de *workloads* conteinerizados em ambientes de nuvem e HPC equipados com AMPs. A abordagem utiliza informações de TLP para determinar ***(i)*** o número adequado de *threads* de cada aplicação e ***(ii)*** o mapeamento de *threads* e *containers* para P-cores e E-cores. Com base em uma etapa de *profiling offline*, o CoreWise orienta o posicionamento em tempo de execução, permitindo a execução concorrente eficiente de múltiplos *containers*, reduzindo o *makespan* e o *Energy-Delay Product* (EDP).

A Figura 1 apresenta uma visão geral do processo de gerenciamento de recursos do CoreWise: inicialmente, o usuário fornece um conjunto de aplicações (**Workloads**) para execução. O CoreWise cria e implanta um *container* para cada aplicação/*workload* (**Containers**), definindo sua configuração de paralelismo (#P, #E) e mapeando as *threads* para cores de desempenho (P) e de eficiência energética (E). Por fim, o gerenciador de

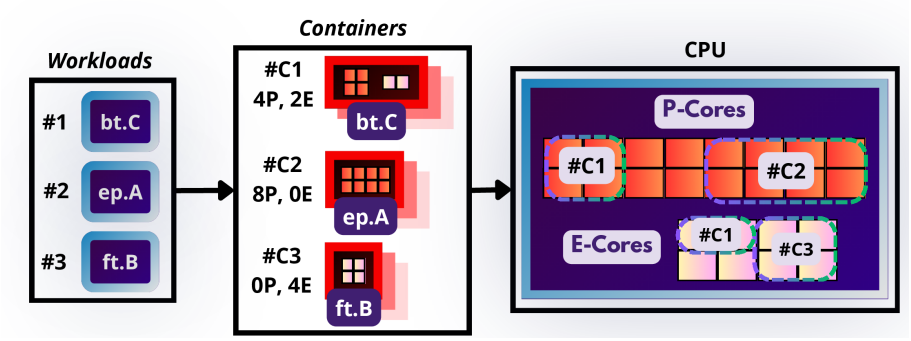


Figura 1. Visão geral do processo de gerenciamento de recursos do CoreWise.

recursos aloca transparentemente um ou mais *containers* ao processador, atribuindo a cada um uma combinação adequada de P-cores e E-cores (**CPU**).

Nossos experimentos comparam o CoreWise com outras sete estratégias considerando 10 *batches* distintos de *workloads*. Para cada *batch*, avaliamos 7 configurações candidatas de número de *threads* em E-cores e P-cores e executamos 50 *containers* concorrentes. Nossos resultados demonstram que o CoreWise supera soluções do estado da arte, incluindo abordagens homogêneas, heurísticas e seriais com afinidade, alcançando uma redução média de 91,2% no EDP em relação ao *baseline* serial padrão, sem penalizar o consumo de energia nem o tempo de execução. Esses resultados destacam a importância do gerenciamento consciente da heterogeneidade e da coordenação global de recursos, particularmente em ambientes com alta carga de *workloads*.

O restante deste artigo está organizado da seguinte forma. A Seção 2 apresenta os conceitos básicos e a Seção 3 discute os trabalhos relacionados sobre AMPs e virtualização. A Seção 4 apresenta uma visão geral da abordagem proposta. A Seção 5 descreve a metodologia experimental, enquanto a Seção 6 apresenta os resultados da avaliação. Por fim, a Seção 7 conclui o artigo.

2. Fundamentação Teórica

Esta seção apresenta os conceitos necessários para compreender os desafios tratados pelo CoreWise: a assimetria entre diferentes tipos de cores, os limites do paralelismo em nível de *thread* e a contenção de recursos em ambientes virtualizados e de nuvem.

Processadores Multicore Assimétricos (AMP). AMPs melhoram a eficiência energética ao combinar cores de alto desempenho (P-cores) e de eficiência energética (E-cores) sob uma arquitetura de conjunto de instruções compartilhada [Saez and Prieto-Matias 2022]. Inicialmente difundidos em dispositivos móveis (ARM DynamIQ/big.LITTLE) [Saez and Prieto-Matias 2022], AMPs também passaram a equipar plataformas *desktop* e servidores, como Apple M4 e Intel Alder Lake [Bilbao et al. 2023]. Embora essa heterogeneidade permita otimizar o compromisso entre desempenho e energia, ela também torna a alocação de recursos em tempo de execução mais complexa.

Computação Paralela em Ambientes Virtualizados e de Nuvem. Aplicações paralelas apresentam demandas distintas de recursos: algumas são *compute-intensive*, enquanto outras são *memory-bound* ou dominadas por sincronização. Embora AMPs possibilitem melhor desempenho e eficiência energética, sua heterogeneidade também impõe novos desafios de execução e escalabilidade, incluindo:

- **Desbalanceamento e sincronização entre tipos de cores:** A assimetria de desempenho entre P-cores e E-cores pode agravar o desbalanceamento de carga, especialmente em *workloads* inerentemente irregulares [Saez and Prieto-Matias 2022]. Pequenas diferenças no tempo de execução das *threads* podem propagar-se e resultar em atrasos em barreiras de sincronização, sobretudo quando P-cores mais rápidos precisam aguardar E-cores mais lentos.
- **Saturação das unidades funcionais:** *Workloads* com alto grau de paralelismo em nível de instrução (*Instruction-Level Parallelism* – ILP) podem alcançar alto desempenho com relativamente poucas *threads* [Lorenzon et al. 2018]. Como AMPs geralmente dispõem de menos P-cores (mais adequados para explorar ILP), a sobrescrição de *threads* computacionalmente intensivas em E-cores pode limitar o *throughput*.
- **Contenção de memória e recursos compartilhados:** Acessos concorrentes à DRAM e a estruturas compartilhadas podem saturar a largura de banda, aumentar o tráfego de coerência e elevar a latência [Suleman et al. 2008, Moori et al. 2025]. Diferenças de capacidade e latência de cache entre os tipos de cores podem agravar esses gargalos e limitar a escalabilidade.

Esses fatores restringem a disponibilidade de instâncias de nuvem baseadas em AMPs para cenários *multitenant*. Como mencionado na Seção 1, as instâncias AWS EC2 Mac (`mac2.metal`, `mac2-m1ultra.metal`, `mac2-m2.metal` e `mac2-m2pro.metal`) operam em modo *bare-metal single-tenant*, o que simplifica o gerenciamento e reduz a interferência em recursos compartilhados.

3. Trabalhos Relacionados

Esta seção organiza os trabalhos relacionados em três linhas: escalonamento e caracterização de AMPs, execução de aplicações paralelas e ajuste de *threads* em nuvem, e gerenciamento de AMPs em ambientes virtualizados e *multitenant*. Finalmente, também discute as lacunas da literatura que motivam a proposta do CoreWise.

Escalonamento e Caracterização de AMPs. Pesquisas recentes concentram-se principalmente em ***(i)*** caracterização microarquitetural [Dalakoti and Chakraborty 2022] e ***(ii)*** escalonamento no nível do sistema operacional [Salami et al. 2022]. Entretanto, esses trabalhos normalmente exploram a heterogeneidade em uma única máquina, sem considerar ambientes *multitenant* baseados em *containers* nem o gerenciamento conjunto do paralelismo em nível de *thread* (TLP) e do mapeamento de *threads* para P-cores e E-cores. Shimchenko et al. [Shimchenko et al. 2024] mostraram que o escalonamento seletivo de tarefas de *garbage collection* reduz o consumo de energia sob restrições de latência. Saez et al. [Saez and Prieto-Matias 2022] avaliaram o Intel Thread Director em processadores Alder Lake, comparando-o a escalonadores assimétricos baseados em contadores de *hardware*. Schöne et al. [Schöne et al. 2024b] caracterizaram o comportamento de desempenho e energia de sistemas Alder Lake considerando mecanismos de gerenciamento de energia, como escalonamento dinâmico de frequência e estados ociosos dos cores.

Ajuste de *Threads* e Aplicações Paralelas em Nuvem. Diversos trabalhos investigam aplicações paralelas em ambientes de nuvem ou técnicas de *thread tuning* (TT) para execuções isoladas [Expósito et al. 2013, Higgins et al. 2015, Zhang and Hoffmann 2016]. Entretanto, essas abordagens geralmente assumem

configurações estáticas, acesso exclusivo aos recursos ou arquiteturas homogêneas, desconsiderando a contenção entre aplicações. Até onde sabemos, apenas Silva et al. [da Silva et al. 2024] exploram explicitamente *multitenancy* em TT, mas sua proposta ainda se restringe a processadores homogêneos.

AMPs em Ambientes Virtualizados e *Multitenant*. Poucos trabalhos investigam a interação entre AMPs e virtualização. Kwon et al. [Kwon et al. 2011b] propõem um *hypervisor* para mapear máquinas virtuais (VM) em cores heterogêneos, enquanto Teabe et al. [Teabe et al. 2017] incorporam consciência de assimetria ao escalonamento de VMs em nuvem. Contudo, ambas as abordagens operam no nível de VMs e não oferecem um gerenciador de recursos para *containers* que decida conjuntamente ***(i)*** o paralelismo em nível de *thread* (TLP), ***(ii)*** o mapeamento para P-cores e E-cores e ***(iii)*** a coordenação de múltiplos *containers* sob contenção de recursos.

3.1. Nossa Contribuição

Em comparação com soluções anteriores para AMPs, que se concentram principalmente em mecanismos de *hardware/kernel* [Saez and Prieto-Matias 2022, Schöne et al. 2024b], gerenciamento da assimetria no nível de VMs [Kwon et al. 2011b, Teabe et al. 2017] ou assumem arquiteturas *multicore* homogêneas para o ajuste do número de *threads* [da Silva et al. 2024], o COREWISE aborda um ponto ainda não explorado: ***o gerenciamento coordenado de recursos, no nível de containers, em arquiteturas com cores heterogêneos***. Especificamente, o COREWISE trata simultaneamente três desafios:

- **Paralelismo em nível de *threads* (TLP):** em vez de assumir um número fixo de *threads*, o COREWISE seleciona e ajusta dinamicamente o grau efetivo de paralelismo de cada *container*, aumentando ou reduzindo o número de *threads* para evitar sobrescrição de recursos (*oversubscription*) e gargalos decorrentes de sincronização e largura de banda de memória.
- **Assimetria entre tipos de cores:** o COREWISE aloca *workloads* considerando os diferentes perfis de desempenho e consumo energético de P-cores e E-cores, evitando desbalanceamentos e impedindo que *threads* críticas para o desempenho sejam executadas em tipos de cores inadequados.
- **Contenção entre múltiplos *containers*:** o COREWISE coordena suas decisões entre *containers* co-localizados, considerando a interferência em recursos compartilhados, em vez de otimizar cada aplicação de forma isolada.

Ao integrar essas três dimensões de controle em um único *pipeline* de decisão em *runtime*, o COREWISE fornece um mecanismo adaptativo que supera as limitações das abordagens existentes, as quais tratam o controle do TLP, o gerenciamento da assimetria e a coordenação entre múltiplos *tenants* como problemas independentes.

Diferentemente de trabalhos anteriores para AMPs, focados em mecanismos de *hardware/kernel* [Saez and Prieto-Matias 2022, Schöne et al. 2024b], VMs [Kwon et al. 2011b, Teabe et al. 2017] ou ajuste de *threads* em arquiteturas homogêneas [da Silva et al. 2024], o COREWISE realiza gerenciamento coordenado de recursos para *containers* em CPUs híbridas. Em *runtime*, o COREWISE adapta o paralelismo em nível de *thread* (TLP), aloca *workloads* considerando a heterogeneidade entre P-cores e E-cores e coordena *containers* co-localizados para minimizar a contenção por recursos compartilhados. Essa integração permite decisões coordenadas de escalonamento que não são contempladas pelas abordagens existentes.

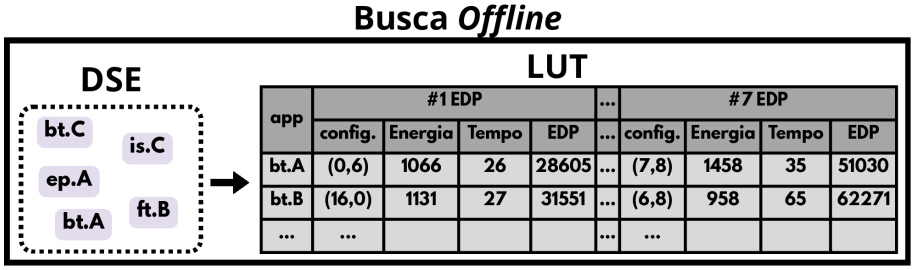


Figura 2. Etapa *Offline*: criação da *Lookup Table* (LUT) de EDP e classificação de configurações.

4. Gerenciador de Recursos COREWISE

O COREWISE é um gerenciador para alocação energeticamente eficiente de *workloads* conteinerizados em sistemas AMP de nuvem/HPC. Ele orquestra a execução concorrente de K containers em uma plataforma com C_P P-cores e C_E E-cores, visando minimizar o *Energy-Delay Product* (EDP) agregado. O gerenciador opera em duas etapas: na *offline*, realiza o *profiling* das aplicações em diferentes configurações de P-cores e E-cores para construir uma *Lookup Table* (**LUT**) ranqueada; na *online*, utiliza essa LUT e o estado da infraestrutura para determinar, em três fases, a alocação de cores aplicada pelo *runtime* de *containers*. As seções seguintes detalham o funcionamento da proposta.

4.1. Etapa *Offline*: *Profiling* e Classificação por EDP

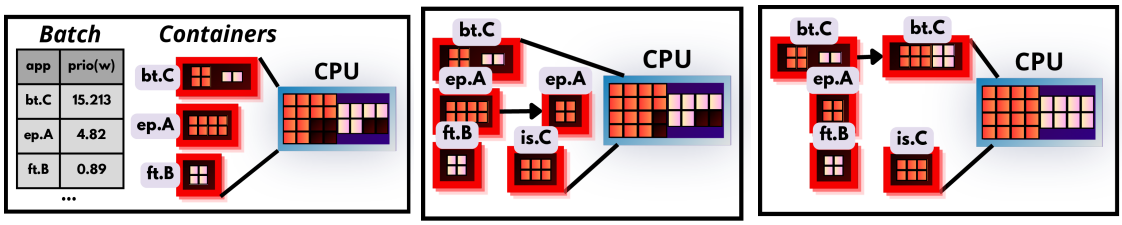
Nesta etapa, cada *workload* é submetido a *profiling offline* (**DSE** de *Design Space Exploration*) em diferentes configurações de cores (p, e), em que p e e representam o número de *threads* executando em P-cores e E-cores, respectivamente. Para cada configuração, são medidos o consumo de energia E (joules)[1], o tempo de execução T (segundos) e o EDP ($EDP = E \times T$). Os resultados são então organizados em uma **LUT**, que associa cada *workload* e configuração (p, e) à tripla (E, T, EDP), ordenada por EDP crescente (#1 EDP … #7 EDP na Figura 2). Em tempo de execução, o gerenciador utiliza apenas as configurações top-N dessa classificação, reduzindo significativamente o espaço de busca em relação à DSE completa (5,51% em nossos experimentos – ver Seção 6). Valores maiores de N aumentam a flexibilidade de alocação, enquanto valores menores priorizam as configurações mais eficientes em EDP.

4.2. Etapa *Online*: *Pipeline* de Três Fases

Dado um lote de K *workloads* ($K = |W|$) e sua LUT, o gerenciador de recursos aloca cores por meio de um *pipeline* iterativo de três fases: *(1)* alocação *best-fit* orientada por prioridade, *(2)* redução cooperativa e *(3)* redistribuição por ganho marginal. O processo se repete até que todos os *workloads* sejam implantados. As Figuras 3a, 3b e 3c ilustram uma iteração desse *pipeline*.

Fase 1: Alocação *Best-Fit* Orientada por Prioridade. Os *workloads* são ordenados por uma pontuação de prioridade $priority(w) = T_w^\star \times \sigma(w)$, em que $T_w^\star$ é o tempo de execução na melhor configuração da LUT e $\sigma(w)$ representa sua sensibilidade de EDP $\sigma(w) = \frac{EDP_w^{(r)}}{EDP_w^{(1)}}, r = \min(5, N)$. Assim, *workloads* mais longos e sensíveis à configuração são priorizados, aumentando a probabilidade de obter suas melhores alocações.

[1]Nesta seção, *E* representa o consumo de energia medido, enquanto E (ou E-) refere-se ao tipo de E-core.



(a) Fase 1. (b) Fase 2. (c) Fase 3.

Figura 3. Etapa *Online*: (a) Fase 1, alocação *best-fit* orientada por prioridade; (b) Fase 2, redução cooperativa; e (c) Fase 3, redistribuição por ganho marginal.

Para cada *workload*, o gerenciador percorre a LUT e seleciona a primeira configuração que satisfaça simultaneamente as restrições de recursos e o fator de tolerância adaptativo $\alpha(w)$: $EDP_w^{(i)} \leq \alpha(w)\, EDP_w^{(1)}$. O fator $\alpha(w)$ varia linearmente com $T_w^\star$:

$$\alpha(w) = \begin{cases} \alpha_{\max} & \text{if } T_w^\star \leq T_{\text{curto}}, \\ \alpha_{\max} + \frac{T_w^\star - T_{\text{curto}}}{T_{\text{longo}} - T_{\text{curto}}} \cdot (1 - \alpha_{\max}) & \text{if } T_{\text{curto}} < T_w^\star < T_{\text{longo}}, \\ 1 & \text{if } T_w^\star \geq T_{\text{longo}}, \end{cases} \tag{1}$$

em que $\alpha_{\max} = 1.18$, $T_{\text{curto}} = 5\,\text{s}$ e $T_{\text{longo}} = 102\,\text{s}$, resultando em $1 \leq \alpha(w) \leq \alpha_{\max}$.

Os parâmetros são obtidos empiricamente a partir do *profiling* (DSE): $\alpha_{\max}$ corresponde à razão média+σ entre o EDP da posição 5 e da posição 1, enquanto T_{curto} e T_{longo} representam os tempos mediano e máximo da melhor configuração. Logo, *workloads* curtos admitem configurações até 18% acima do EDP ótimo, enquanto *workloads* longos são restritos às melhores configurações. Caso nenhuma configuração aceitável esteja disponível, o *workload* é adiado para a próxima iteração.

A Figura 3a ilustra uma iteração da Fase 1, mostrando a alocação do *batch* composto pelos *workloads* bt.C, ep.A e ft.B de acordo com a política de prioridade proposta. Ao final dessa etapa, os respectivos *containers* são mapeados aos recursos computacionais conforme os passos descritos anteriormente.

Fase 2: Redução Cooperativa. Para cada *workload* adiado d, o gerenciador verifica se reduzir um *workload* já alocado (*doador*) para uma configuração com menos cores libera recursos suficientes para escalonar d. A decisão compara o EDP da execução *sequencial*, em que ambos executam em suas configurações preferidas, com o da execução *concorrente*, em que o doador utiliza uma configuração reduzida. Na Figura 3b, por exemplo, ep.A é reduzido para permitir a execução de is.C.

Considere um doador executando na configuração c, reduzido para c' ($p_{c'} \leq p_c$ e $e_{c'} \leq e_c$), liberando recursos para que d utilize a configuração c_d. O gerenciador calcula: $EDP_{\text{seq}} = E_{\text{doador}}^{(c)} T_{\text{doador}}^{(c)} + E_d^{(1)} T_d^{(1)}$ e $EDP_{\text{con}} = \left(E_{\text{doador}}^{(c')} + E_d^{(c_d)}\right) \max\left(T_{\text{doador}}^{(c')}, T_d^{(c_d)}\right)$, em que, no EDP da execução concorrente (EDP_{con}), as energias são somadas e o tempo corresponde ao maior tempo de execução. O ganho líquido da troca é $\Delta = EDP_{\text{seq}} - EDP_{\text{con}}$. A troca é realizada apenas se $\Delta > 0$, selecionando-se a combinação (*doador*, c', d, c_d) com maior ganho. O processo é repetido até que não existam mais trocas benéficas.

Fase 3: Redistribuição por Ganho Marginal. Após as Fases 1 e 2, alguns P-

cores e E-cores podem permanecer ociosos. O gerenciador os redistribui iterativamente, selecionando o *upgrade* viável que maximiza o ganho de EDP por core adicional. A Figura 3c ilustra esse processo, no qual bt.C tem *upgrade*. Para cada *workload* que ainda não esteja na configuração da posição 1, o gerenciador avalia todos os *upgrades* possíveis e calcula $g(w, i \to j) = \frac{EDP_w^{(i)} - EDP_w^{(j)}}{(p_j - p_i) + (e_j - e_i)}$, com $j < i$, $p_i \leq p_j$ e $e_i \leq e_j$, em que i e j representam, respectivamente, as configurações atual e candidata. O *upgrade* com maior g recebe os cores disponíveis, repetindo-se o processo até que não existam *upgrades* viáveis.

4.3. Transparência e Implantação

COREWISE opera inteiramente no nível de orquestração de *containers*, sendo transparente às aplicações. Não requer modificações nas aplicações, imagens de *containers* ou no sistema operacional convidado. As decisões de escalonamento são implementadas externamente por mecanismos do *runtime* de *containers*, como afinidade de CPUs (`cpuset_cpus`) e variáveis de ambiente (`OMP_NUM_THREADS` e `GOMP_CPU_AFFINITY`).

5. Metodologia Experimental

Esta seção descreve a metodologia empregada em nossos experimentos, abrangendo o ambiente computacional, os *workloads* avaliados, a métrica de eficiência energética, o espaço de busca do *profiling offline* e as estratégias comparadas.

Configuração do Ambiente. Os experimentos foram executados em uma workstation com um processador Intel Core i9-12900K (Alder Lake), Ubuntu 22.04 LTS e kernel Linux 5.15. O processador possui 8 P-cores (*16 threads com Hyper-Threading*), 8 E-cores organizados em dois *clusters*, caches L1 privadas de 32 KiB, caches L2 de 1.25 MiB por P-core e 2 MiB por cluster de E-cores, cache L3 compartilhada de 30 MiB e 32GiB de memória DDR5.

Conjunto de *Workloads*. Consideramos nove *kernels* do NAS *Parallel Benchmarks* (NPB) [Bailey et al. 1991], versão 3.3, compilados com GCC 11.4 e OpenMP, que cobrem um amplo espectro de características computacionais: *Compute-bound* – BT (*Block Tridiagonal*), SP (*Scalar Pentadiagonal*), LU (*Lower-Upper symmetric Gauss-Seidel*); *Memory-bound* – CG (*Conjugate Gradient*), MG (*Multigrid*), FT (*Fourier Transform*), IS (*Integer Sort*); Altamente paralelizável – EP (*Embarrassingly Parallel)*; e Não estruturado – UA (*Unstructured Adaptive*). Cada *kernel* foi compilado em três tamanhos de problema (classes A, B e C), resultando em 27 *workloads* distintos, cujos tempos de execução variam de aproximadamente 5 s a mais de 100 s na plataforma-alvo.

Métrica Avaliada de Eficiência Energética. Avaliamos o EDP [Gonzalez and Horowitz 1996], definido como $\mathrm{EDP} = E \times T$, que captura o compromisso entre consumo de energia (E) e desempenho (tempo de execução, T). O tempo T corresponde ao intervalo entre o lançamento do primeiro *container* (ou *workload* conteinerizado) e a conclusão do último *workload* do lote. A energia E foi medida com `perf stat -e power/energy-pkg/`, utilizando o contador Intel Running Average Power Limit (RAPL) [David et al. 2010] do pacote da CPU.

***Profiling Offline* de EDP.** Antes dos experimentos, cada *workload* foi submetido a *profiling* em todas as combinações de contagem de *threads* e alocação de cores (Seção 4.1). Cada configuração é representada por uma tupla (p, e), em que $p \in$

Tabela 1. Resumo das estratégias avaliadas.

Estratégia	Consciência de AMP	Seleção de Cores	TLP	Coordenação	Escalonamento
CWISE	**Consciente**	P/E	**Adaptativo**	**Coordenada**	**LUT baseada em EDP**
CWISE (HOMOGEN)	Agnóstica	P/E	Adaptativo	Coordenada	LUT baseada em EDP
RR	Agnóstica	P/E	Fixo ($T = 24$)	Compartilhada	Round-robin ($q = 5$ s)
SJF	Agnóstica	P/E	Fixo ($T = 24$)	Compartilhada	Menor tarefa primeiro
BL	Agnóstica	P/E	Fixo ($T = 24$)	Nenhuma	Serial
BL (P)	Consciente	P	Fixo ($T = 16$)	Nenhuma	Serial
BL (E)	Consciente	E	Fixo ($T = 8$)	Nenhuma	Serial
BL (ALT P/E)	Consciente	P/E alternado	Fixo ($T = 16$ ou 8)	Nenhuma	Serial

$\{0, \ldots, 16\}$ e $e \in \{0, \ldots, 8\}$, com $p + e > 0$, totalizando 152 configurações por *workload*.

Estratégias Avaliadas. Realizamos uma avaliação por ablação do COREWISE (denominado CWISE nos gráficos), comparando baselines que isolam ***(i)*** o impacto do mapeamento consciente da heterogeneidade, ***(ii)*** a eficácia da alocação adaptativa frente a heurísticas de *time-sharing* sob sobrescrição e ***(iii)*** o impacto de políticas de afinidade serial. Todas as estratégias executaram os *workloads* em *containers* Docker, com afinidade via `--cpuset-cpus`, quando aplicável, e `OMP_NUM_THREADS` e `GOMP_CPU_AFFINITY` configurados conforme a alocação. Os experimentos consideraram alta contenção, com N=7 configurações candidatas, 10 *batches* randomizados e K=50 *containers* concorrentes. O valor de N foi definido empiricamente, pois sua variação entre 3 e 8 produziu apenas alterações marginais, com coeficiente de variação de $0,43\%$.

As estratégias e suas características são resumidas na Tabela 1. Consciência de AMP indica se a estratégia considera a heterogeneidade; Seleção de Cores define os tipos de cores utilizados; TLP indica paralelismo Fixo ou Adaptativo; Coordenação especifica se as alocações são Coordenadas, Compartilhadas ou Nenhuma; e Escalonamento define o critério de ordenação ou alocação dos *workloads*. As estratégias avaliadas são:

- **COREWISE (HOMOGEN).** Uma variante homogênea do COREWISE, baseada nos mesmos princípios do trabalho de Silva et al. [da Silva et al. 2024], que desconsidera a assimetria de desempenho ao substituir as alocações (p, e) por uma única configuração de número de *threads* t. Cada *container* é fixado a quaisquer t CPUs lógicas livres, sem distinção entre P-cores e E-cores, utilizando `OMP_NUM_THREADS`$= t$. O *profiling offline* é análogo ao do COREWISE, porém considera apenas um único parâmetro de contagem de *threads*, $t \in 1, \ldots, 24$.
- **RR (ROUND-ROBIN).** Um *baseline* que inicia um *container* por *workload* e fixa todos os *containers* ao mesmo conjunto compartilhado de CPUs (*cpuset*), composto por todas as CPUs lógicas disponíveis no início da execução. Os *containers* são inicialmente pausados e executados em rotação *First-In First-Out* (FIFO) estrita, sendo cada um reativado por um quantum fixo ($q = 5$s, por padrão) antes de ser novamente pausado e reinserido na fila, caso sua execução ainda não tenha sido concluída.
- **SJF (SHORTEST-JOB-FIRST).** Um *baseline Shortest-Job-First* (SJF) que não utiliza a LUT ranqueada pelo EDP nem particiona os cores entre os *containers*. Em vez disso, co-localiza todo o *batch*, fixando cada *container* ao mesmo *cpuset* completo (todas as CPUs lógicas disponíveis) e configurando `OMP_NUM_THREADS` de acordo com o tamanho dessa máscara compartilhada.

- **BL (Default).** Um *baseline* serial naïve que executa os *workloads* sequencialmente, sem especificar uma máscara explícita de CPUs (isto é, cada *container* pode executar em qualquer CPU lógica disponível), utilizando `OMP_NUM_THREADS`$= 24$.
- **BL (P), BL (E) e BL (ALT P/E).** Três *baselines* seriais que executam o *batch* sequencialmente, fixando cada *container* a conjuntos fixos de cores: **BL (P)** fixa cada *workload* às CPUs lógicas dos P-cores $0, \ldots, 15$, utilizando `OMP_NUM_THREADS`$= 16$; **BL (E)** fixa cada *workload* às CPUs dos E-cores $16, \ldots, 23$, utilizando `OMP_NUM_THREADS`$= 8$; e **BL (ALT P/E)** alterna essas duas máscaras de afinidade entre *workloads* consecutivos (índice par $\rightarrow$ P, índice ímpar $\rightarrow$ E).

Todos os experimentos foram realizados em uma plataforma própria para emular ambientes HPC/nuvem. Assim, os resultados devem ser considerados otimistas em relação a plataformas comerciais AMP, como as instâncias AWS EC2 Mac [aws 2024], que exigem *hosts* dedicados com alocação mínima de 24 horas, o que torna inviável reproduzir integralmente nossos experimentos.

6. Resultados

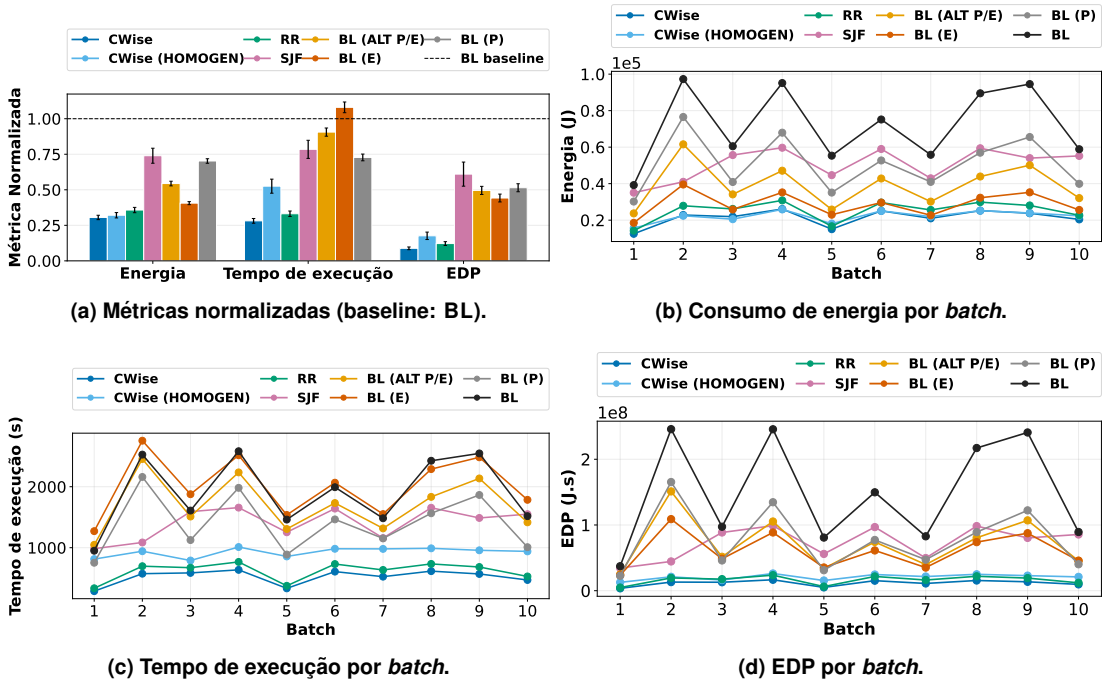
Esta seção apresenta os resultados experimentais, comparando as estratégias tanto em termos do desempenho geral em tempo de execução, consumo de energia e EDP quanto dos resultados obtidos para cada *batch* executado.

6.1. Resultados Gerais

Esta seção apresenta os resultados gerais obtidos pela aplicação da Etapa *Online* do CoreWise (CWise nos gráficos) – Seção 4.2. Para isso, a Figura 4a resume as métricas médias normalizadas (Energia, Tempo de Execução e EDP – conjuntos de barras no eixo *y*) com barras de erro padrão (*Standard Error of the Mean* – SEM), considerando todos os 10 *batches* diferentes executados pelas estratégias avaliadas (eixo *x*). Elas são normalizadas pelo *baseline* serial padrão BL (valores menores que 1 indicam melhoria).

Melhorias de EDP. O CoreWise alcançou o menor EDP, com EDP médio normalizado de $0,088$ (SEM $= 0,010$, Intervalo de Confiança – IC de 95% $[0,069; 0,107]$), correspondendo a uma redução de EDP de $91,2\%$ em relação ao BL. Entre as demais estratégias, RR teve o melhor desempenho (redução de $87,8\%$), mas ainda permaneceu $27,86\%$ pior que o CoreWise. Essa melhoria é consequência da localidade temporal de cache: ao preservar a afinidade de core (e, portanto, reutilizar caches aquecidas), o RR pode reduzir penalidades de aquecimento de cache sob alta sobrecarga, com $45,94\%$ menos misses de carga na cache de último nível (LLC) em comparação com o CoreWise. O SJF apresentou EDP substancialmente maior (apenas 39% de redução em relação ao BL), indicando que selecionar jobs curtos sem atribuição de cores consciente da heterogeneidade é insuficiente sob alta sobrecarga.

Decomposição de energia e tempo. Os ganhos de EDP foram impulsionados por melhorias em ambas as dimensões (tempo de execução e consumo de energia). O CoreWise reduziu a energia em $69,5\%$ e o tempo de execução em $71,9\%$. O RR alcançou médias competitivas, mas ainda inferiores ($7,62\%$ mais energia e $6,95\%$ mais tempo de execução), confirmando que mesmo uma alocação cíclica simples se beneficia do paralelismo entre *containers*, mas ainda fica aquém do CoreWise.



(a) Métricas normalizadas (baseline: BL).

(b) Consumo de energia por *batch*.

(c) Tempo de execução por *batch*.

(d) EDP por *batch*.

Figura 4. Resultados para $N = 7$ e $K = 50$: (a) métricas normalizadas (baseline: BL), (b) consumo de energia por *batch*, (c) tempo de execução por *batch* e (d) EDP por *batch*.

Importância da consciência de assimetria. A variante CoreWise (HOMOGEN) consumiu uma quantidade semelhante de energia ($3,42\%$ a mais), mas executou significativamente mais lentamente ($46,66\%$) que o CoreWise, produzindo um EDP médio normalizado de $0,176$. Em outras palavras, ignorar a heterogeneidade de cores aproximadamente dobrou o EDP em comparação com o projeto consciente de assimetria, demonstrando que o mapping correto de *workloads* para recursos de P-/E-core é um fator relevante nas melhorias de EDP observadas.

Baselines seriais com fixação de core. Todas as variantes seriais fixadas melhoraram em relação à referência BL naïve, mas permaneceram distantes do CoreWise. A melhor variante serial em termos de EDP foi a BL (E), com redução de $57,8\%$, refletindo menor energia (redução de $59,4\%$) ao custo de maior tempo de execução (aumento de 8%). Entretanto, a BL (P), apesar de reduzir tanto a energia quanto o tempo de execução, ainda apresentou ganhos de EDP menores que as variantes BL (E) e BL (ALT P/E). Por fim, esses resultados indicam que políticas simples de afinidade podem mitigar parcialmente o desperdício de energia, mas não podem substituir o planejamento consciente de concorrência em ambientes sobrecarregados.

6.2. Análise por *Batch*

Agora apresentamos os resultados considerando a execução por *batch*. As Figuras 4b, 4c e 4d fornecem uma visão semelhante à da seção anterior, mas a análise agora é desagregada por *batch* de aplicações (os valores no eixo x correspondem aos IDs dos *batches* avaliados). Os gráficos apresentam os valores brutos de todas as métricas avaliadas.

Os gráficos por *batch* mostram que o CoreWise permaneceu consistentemente melhor que todas as alternativas nos 10 *batches*. Quantitativamente, seus ganhos de

EDP variaram de reduções de $94,7\%$ a $86,7\%$, sugerindo que a vantagem da nossa proposta persiste em diversos *batches* gerados aleatoriamente. Vale notar também que vários *batches* (por exemplo, os índices 2, 4 e 9) são significativamente piores em todas as variantes BL em energia, tempo de execução e EDP, como mostrado nos gráficos. Esses padrões são facilmente percebidos como um deslocamento no eixo *y*, o que pode sugerir uma piora proporcional e fixa na escalabilidade paralela de algumas aplicações selecionadas aleatoriamente para cada *batch*.

6.3. Discussão

Os resultados produzem três *insights* principais. ***Primeiro, o gerenciamento de recursos consciente de EDP gera grandes ganhos sob contenção***: para $N{=}7$ e $K{=}50$, o COREWISE reduziu o EDP em $91,22\%$ em média em comparação com o BL, melhorando simultaneamente a energia (redução de $69,5\%$) e o tempo de execução (redução de $71,9\%$). Isso mostra que nossa técnica não melhora uma métrica à custa de outra, mas reduz simultaneamente o consumo de energia e melhora o tempo de execução. ***Segundo, a consciência de AMP importa.*** A versão homogênea, COREWISE (HOMOGEN), alcançou energia semelhante, mas foi muito mais lenta, dobrando o EDP em relação ao COREWISE. Isso destaca que, em AMPs, a escolha das alocações de P-/E-core é tão importante quanto a decisão de executar *containers* concorrentemente. ***Por fim, as estratégias comparadas generalizam mal para AMP.*** Suas políticas fixas não consideram as capacidades heterogêneas dos cores nem a variabilidade das aplicações paralelas sob sobrecarga, levando a posicionamentos subótimos. Embora o RR tenha sido o concorrente não COREWISE mais forte, ele ainda resultou em EDP 38% maior que o COREWISE. As variantes SJF e seriais fixadas alcançaram melhorias parciais em relação ao BL, mas foram várias vezes piores em todas as métricas de comparação com o COREWISE.

7. Conclusão

Este artigo apresentou o COREWISE, um gerenciador de recursos para execução energeticamente eficiente de *workloads* conteinerizados em sistemas AMPs. Ao combinar *profiling offline* de EDP com um *pipeline* de alocação em *runtime* de três fases, ele mapeia eficientemente aplicações executadas em Docker para P-cores e E-cores. Nossos experimentos em um Intel Core i9-12900K sob alta sobrecarga mostraram que o COREWISE reduziu o EDP médio por *batch* em até 91,2% em relação ao *baseline* serial e superou consistentemente estratégias clássicas de escalonamento de *containers*, alcançando melhorias de até 85,54%. Esses resultados demonstram que o gerenciamento de recursos consciente de assimetria é um habilitador fundamental da execução energeticamente eficiente de *containers* em plataformas de nuvem e edge de próxima geração. Trabalhos futuros incluem substituir a etapa *offline* por uma estratégia leve, bem como integrar o COREWISE ao Kubernetes/Apptainer para o gerenciamento de recursos de cores assimétricos em múltiplos nós.

Agradecimentos

Esse estudo foi financiado em parte pela CAPES - Finance Code 001, pela FAPERGS e pelo CNPq.

Referências

(2024). Amazon ec2 mac instances. https://docs.aws.amazon.com/AWSEC2/la-test/UserGuide/ec2-mac-instances.html. Accessed: 2026-02-12.

Bailey, D. H., Barszcz, E., Barton, J. T., et al. (1991). The NAS parallel benchmarks. *Int. J. High Perform. Comput. Appl.*, 5(3):63–73.

Balla, D., Simon, C., and Maliosz, M. (2020). Adaptive scaling of Kubernetes pods. In *Proceedings of the IEEE/IFIP Network Operations and Management Symposium (NOMS)*, pages 1–5. IEEE.

Bilbao, C., Saez, J. C., and Prieto-Matias, M. (2023). Flexible system software scheduling for asymmetric multicore systems with pmcsched: A case for intel alder lake. *CCPE*, page e7814.

da Silva, V. S., de Lima, E. C., Schwarzrock, J., Rossi, F. D., Luizelli, M. C., Beck, A. C. S., and Lorenzon, A. F. (2024). Synergistically rebalancing the edp of container-based parallel applications. *IEEE Transactions on Parallel and Distributed Systems*, 35(3):484–498.

Dalakoti, V. and Chakraborty, D. (2022). Apple m1 chip vs intel (x86). *IJRD*, 7(5):207–211.

David, H., Gorbatov, E., Hanebutte, U. R., Khanna, R., and Le, C. (2010). Rapl: Memory power estimation and capping. In *Proceedings of the 16th ACM/IEEE international symposium on Low power electronics and design*, pages 189–194.

Expósito, R. R., Taboada, G. L., Ramos, S., Touriño, J., and Doallo, R. (2013). Performance analysis of hpc applications in the cloud. *Future Generation Computer Systems*, 29(1):218–229.

Gonzalez, R. and Horowitz, M. (1996). Energy dissipation in general purpose microprocessors. *IEEE J. Solid-State Circuits*, 31(9):1277–1284.

Higgins, J., Holmes, V., and Venters, C. (2015). Orchestrating docker containers in the hpc environment. In *International Conference on High Performance Computing*, pages 506–513. Springer.

Kwon, Y., Kim, C., Maeng, S., and Huh, J. (2011a). Virtualizing performance asymmetric multi-core systems. In *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*, pages 45–56, New York. ACM.

Kwon, Y., Kim, C., Maeng, S., and Huh, J. (2011b). Virtualizing performance asymmetric multi-core systems. *ACM SIGARCH Computer Architecture News*, 39(3):45–56.

Laguna, F. N., Moori, M. K., Rocha, H. M. G. D. A., and Beck, A. C. S. (2026). Energy-aware tuning of pthreads and openmp applications for asymmetric multicore architectures. In *2026 39th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI)*, pages 1–5. IEEE.

Liu, P. and Guitart, J. (2022). Performance characterization of containerization for hpc workloads on infiniband clusters: an empirical study. *Cluster Computing*, 25(2):847–868.

Lorenzon, A. F., De Oliveira, C. C., Souza, J. D., and Beck, A. C. S. (2018). Aurora: Seamless optimization of openmp applications. *TPDS*, 30(5):1007–1021.

Moori, M. K., Lorenzon, A. F., Rocha, H. M. G. d. A., and Beck, A. C. S. (2025). Gaamp: Automatic thread count/affinity and dvfs tuning for asymmetric multicores. In *2025 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, volume 1, pages 1–6.

Saez, J. C. and Prieto-Matias, M. (2022). Evaluation of the intel thread director technology on an alder lake processor. In *ACM SIGOPS APSYS*, pages 61–67.

Salami, B., Noori, H., and Naghibzadeh, M. (2022). Online energy-efficient fair scheduling for heterogeneous multi-cores considering shared resource contention. *The Journal of Supercomputing*, pages 1–20.

Schöne, R., Ilsche, T., Bielert, M., Gocht-Zech, A., and Hackenberg, D. (2024a). Energy efficiency features of the Intel Alder Lake architecture. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 146–155. IEEE.

Schöne, R., Velten, M., Hackenberg, D., and Ilsche, T. (2024b). Energy efficiency features of the intel alder lake architecture. In *ICPE*, pages 95–106.

Shimchenko, M., Österlund, E., and Wrigstad, T. (2024). Scheduling garbage collection for energy efficiency on asymmetric multicore processors. *arXiv preprint arXiv:2403.02200*.

Suleman, M. A., Qureshi, M. K., and Patt, Y. N. (2008). Feedback-driven threading: power-efficient and high-performance execution of multi-threaded workloads on cmps. *ACM Sigplan Notices*, 43(3):277–286.

Teabe, B., Wapet, P. L., Tchana, A., and Hagimont, D. (2017). Dealing with performance unpredictability in an asymmetric multicore processor cloud. In *European Conference on Parallel Processing*, pages 332–344. Springer.

Zhang, H. and Hoffmann, H. (2016). Maximizing performance under a power cap: A comparison of hardware, software, and hybrid techniques. *ACM SIGPLAN Notices*, 51(4):545–559.