UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CIÊNCIA DA COMPUTAÇÃO

Nicolas Vanz

Virtualização e Migração de Processos em um Sistema Operacional Distribuído para Lightweight Manycores

Florianópolis
25 de julho de 2022

RESUMO

A classe de processadores *lightweight manycore* surgiu para prover um alto grau de paralelismo e eficiência energética. Contudo, o desenvolvimento de aplicações para esses processadores enfrenta diversos problemas de programabilidade provenientes de suas peculiaridades arquitetônicas. Especialmente, o gerenciamento de processos precisa mitigar problemas provenientes das pequenas memórias locais e da falta de um suporte robusto para virtualização. Nesse contexto, este trabalho visa desenvolver o suporte da migração de processos em um Sistema Operational (SO) distribuído para *lightweight manycores* através de uma abordagem de virtualização leve baseada em contêineres. Particularmente, este trabalho está incluído no projeto Nanvix, um SO distribuído de código aberto projetado para *lightweight manycores*. Ao final deste trabalho espera-se melhorar o gerenciamento de processos no Nanvix, bem como abstrair e auxiliar o gerenciamento dos recursos do processador.

Palavras-chave: lightweight manycores. sistemas operacionais. migração de processos. virtualização. conteinerização

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

Durante muitos anos, o aumento do desempenho de sistemas computacionais esteve intrinsecamente associado ao aumento da frequência de relógio dos processadores e avanços na tecnologia dos semicondutores. Essas técnicas se mantiveram eficientes até o momento em que a dissipação de calor interna dos *chips* necessária para viabilizar o aumento da frequência atingiu um limite físico. Este fato associado com o fim iminente da lei de Moore (MOORE, 1965), fez com que a exploração de novas maneiras de aumentar o poder computacional do sistemas se tornasse uma prioridade.

Como alternativa à limitação do aumento da frequência de relógio, foram desenvolvidos processadores com vários núcleos de processamento, os *multicores*. O desempenho dos processadores *multicore* não dependem mais apenas das altas frequências de relógio, recorrendo ao paralelismo como principal vantagem aos processadores *single-core*. Deste modo, mesmo com a estagnação da frequência de relógio nos processadores, esse aumento na quantidade de *cores* em conjunto com outras melhorias no *hardware*, como o aumento no número de transistores nos *chips*, aperfeiçoamento dos preditores de desvio e adaptações na hierarquia de memória, o desempenho dos sistemas computacionais continuaram a aumentar.

Contudo, a eficiência energética dos sistemas computacionais revelou-se tão importante quanto seu desempenho. Segundo o Departamento de Defesa do Governo dos Estados Unidos (DARPA/IPTO) (KOGGE et al., 2008), a potência recomendada para um supercomputador atingir o *exascale* (10^{18} *Floating-point Operations per Second* (FLOPS)), é de 20 MW, o que é inviável para a realidade dos sistemas computacionais modernos. Nesse cenário, observou-se o surgimento de uma nova classe dos processadores denominada *lightweight manycores*. Esses processadores são classificados como Multiprocessor System-on-Chips (MPSoCs) e tem como objetivo atrelar alto desempenho à eficiência energética (FRANCESQUINI et al., 2015). Para atingir esse objetivo, a arquitetura dos *lightweight manycores* é caracterizada por:

(i) Integrar de centenas à milhares de núcleos de processamento operando a baixas frequências em um único *chip*;
(ii) Processar cargas de trabalho *Multiple Instruction Multiple Data* (MIMD);
(iii) Organizar os núcleos em conjuntos, denominados *clusters*, para compartilhamento de recursos locais;
(iv) Utilizar *Networks-on-Chip* (NoCs) para transferência de dados entre núcleos ou *clusters*;
(v) Possuir sistemas memória distribuída restritivos, compostos por pequenas memórias locais; e
(vi) Apresentar componentes heterogêneos.

A Figura 1 ilustra uma visão conceitual da arquitetura de um *lightweight manycore*. Os

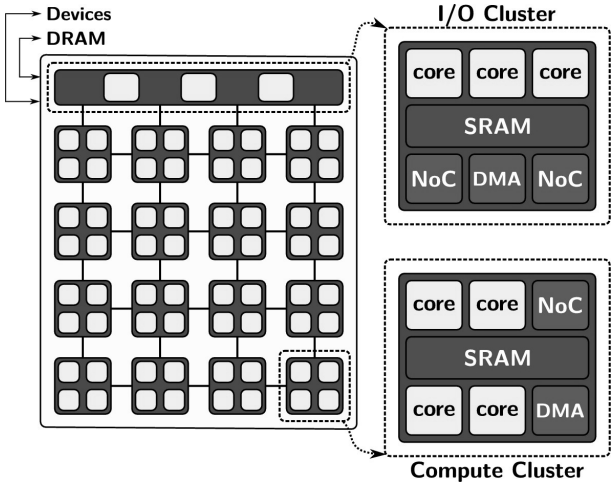


Figura 1 – Visão conceitual de um processador *lightweight manycore* (PENNA et al., 2021)

processadores Kalray MPPA-256 (DINECHIN et al., 2013), PULP (ROSSI et al., 2017) e Sunway SW26010 (FU et al., 2016) são exemplos comerciais dessa classe de processadores.

Apesar dos processadores *lightweight manycore* serem uma alternativa às abordagens tradicionais no que se refere ao aumento de desempenho, as características arquiteturais introduzem severos problemas de programabilidade ao desenvolvimento de *software* de aplicações paralelas (CASTRO et al., 2016). Entre eles, podemos citar:

(i) Necessidade do uso de um modelo de programação híbrida que força troca de informação entre os *clusters* exclusivamente por troca de mensagens via NoC enquanto a comunicação interna em um *cluster* ocorre sobre memória compartilhada (KELLY; GARDNER; KYO, 2013);
(ii) Presença de um sistema de memória distribuída restritivo, formado por múltiplos espaços de endereçamento, exige o particionamento do conjunto de dados em blocos pequenos para manipulação nas pequenas memórias locais. A manipulação deve ocorrer um bloco de cada vez, necessitando a troca explicita de blocos com uma memória remota (CASTRO et al., 2016);
(iii) Maiores latências e gargalos de comunicação através da NoC comparado com comunicação em memória compartilhada;
(iv) Falta de suporte de coerência de cache em *hardware* para economia de energia, exiging do programador a gerência da cache via *software*; e
(v) Configuração heterogênea do *hardware*, como *clusters* destinados a funcionalidades específicas (computação útil e I/O), dificulta o desenvolvimento de aplicações.

Atualmente, estudos exploram soluções para amenizar o impacto das arquiteturais sobre o desenvolvimento de *software*. Sistemas Operacionais (SOs) distribuídos sobresaem-se por proverem um ambiente de programação mais robusto e rico (ASMUSSEN et al., ; KLUGE; GERDES; UNGERER, ; PENNA et al., 2019). Dentre essas soluções, o modelo de um SO distribuído baseado em uma abordagem *multikernel* destaca-se

por aderir a natureza distribuída e restritiva dos *lightweight manycores* (PENNA et al., 2017; PENNA et al., 2017; PENNA et al., 2019).

Nesse cenário, a virtualização dos recursos do processador é importante para o suporte a multi-aplicação e melhor uso do *hardware* disponível (VANZ; SOUTO; CASTRO, 2022). Contudo, as características arquiteturais dos *lightweight manycores*, especialmente relacionadas à memória, inviabilizam um suporte complexo para virtualização. Por exemplo, máquinas virtuais utilizadas em ambientes *cloud* possuem à disposição centenas de GBs de memória para isolar duplicatas inteiras do SO com a ajuda de virtualização a nível de instrução (SHARMA et al., 2016). Nos *lightweight manycores*, as pequenas memórias locais e a simplificação do *hardware* para reduzir o consumo energético restringem os tipos de virtualização suportados.

Neste contexto, este trabalho explora um modelo mais leve de virtualização para *lightweight manycores* baseado no conceito de contêineres. Contêineres são executados pelo SO como aplicações virtuais e não incluem um SO convidado, resultando em um menor impacto no sistema de memória e requisitando menor complexidade do *hardware* (THALHEIM et al., 2018; SHARMA et al., 2016).

1.1 OBJETIVOS

Com base nas motivações citadas previamente, os objetivos deste trabalho serão especificados nas próximas seções.

1.1.1 Objetivo Principal

O objetivo principal deste trabalho é virtualizar os recursos internos de um *cluster* de um *lightweight manycore*. Ao desvincular os recursos locais utilizados por um processo dentro do Nanvix, um SO para *lightweight manycores*, nós conseguiremos prover maior controle e mobilidade de processos no processador.

1.1.2 Objetivos Específicos

(i) Propor um modelo de virtualização adaptado às necessidades e restrições dos *lightweight manycores*;
(ii) Implementar o modelo proposto no Nanvix, um SO distribuído para *lightweight manycores*;
(iii) Analisar a corretude da solução através do desenvolvimento de *benchmarks* que avaliem a migração de processos;
(iv) Analisar o impacto do modelo de virtualização nos diversos níveis de abstração do Nanvix.

1.2 CONTRIBUIÇÕES

Esse trabalho de conclusão propõe o suporte à virtualização e migração de processos no Nanvix. Parte desse trabalho foi publicado na Escola Regional de Alto Desempenho da Região Sul (ERAD/RS) e recebeu o prêmio Aurora Cera de melhor artigo do Fórum de Iniciação Científica (VANZ; SOUTO; CASTRO, 2022).

1.3 ORGANIZAÇÃO DO TRABALHO

Os próximos capítulos do trabalho estão organizadas da seguinte maneira. O Capítulo 2 apresenta conceitos fundameitais para o entendimento do trabalho, tais como um detalhamento da arquitetura dos *lightweight manycores* e do Nanvix. O Capítulo 3 discute os trabalhos relacionados. O Capítulo 4 expõe a proposta deste trabalho de conclusão e os detalhes de desenvolvimento da solução a ser implementada. O Capítulo 5 exibe e discute alguns resultados preliminares já obtidos. Por fim, o Capítulo 6 apresenta as conclusões deste trabalho e pontua os próximos passos da pesquisa.

2 REFERENCIAL TEÓRICO

Neste capítulo serão apresentados conceitos fundamentais para o entendimento do trabalho. A Seção 2.1 apresenta uma visão geral da evolução dos processadores, partindo dos *single-cores* até os *lightweight manycores*. A Seção 2.2 apresenta o Nanvix, SO que será utilizado no desenvolvimento deste trabalho. A Seção 2.3 descreve detalhes importantes sobre a virtualização e migração de processos.

2.1 DOS *SINGLE-CORES* AOS *LIGHTWEIGHT MANYCORES*

O aumento de desempenho dos sistemas computacionais manteve-se como uma necessidade constante para o avanço da ciência em vários setores: astrologia, biologia, engenharia, etc. Até tempos atrás, esse objetivo era alcançado através do aumento da frequência de relógios do núcleo de processamento, do avanço na tecnologia dos semicondutores e do acréscimo do número de transistores em um *chip*. Atualmente, nós estamos chegando ao limite físico que impede a aplicação de parte dessas técnicas. Além da dificuldade de garantir a dissipação de calor à medida que a frequência aumenta, o número de transistores que conseguimos colocar em uma mesma área de um *chip* está chegando ao seu limite físico, i.e., o tamanho dos transitores alcançou a escala atômica.

Como alternativa para a continuidade nos avanços de poder computacional, foram exploradas novas técnicas. Em especial, foram desenvolvidas arquiteturas paralelas, que exploram o poder de processamento paralelo, o qual é atingido pela execução de múltiplos *cores* simultaneamente. Essas novas arquiteturas são classificadas de acordo com a maneira com que conseguem manipular os dados. São elas: (i) Single Instruction Single Data (SISD); (ii) Single Instruction Multiple Data (SIMD); (iii) Multiple Instruction Single Data (MISD); (iv) MIMD. Neste trabalho, estamos interessados nas arquiteturas que suportam cargas de trabalho MIMD, as quais ainda podem ser divididas em multiprocessadores ou multicomputadores, como mostrado na Figura 2 (TANENBAUM; BOS, 2014).

Neste contexto, a classe de processadores *lightweight manycores* destacam-se por atrelar alto poder de processamento paralelo com eficiência energética. Os *lightweight manycores* são classificados como MPSoC e suas arquiteturas apresentam as seguintes características:

(i) Integrar de centenas à milhares de núcleos de processamento operando a baixas frequências em um único *chip*;
(ii) Processar cargas de trabalho MIMD;
(iii) Organizar os núcleos em conjuntos, denominados *clusters*, para compartilhamento de recursos locais;
(iv) Utilizar NoCs para transferência de dados entre núcleos ou *clusters*;

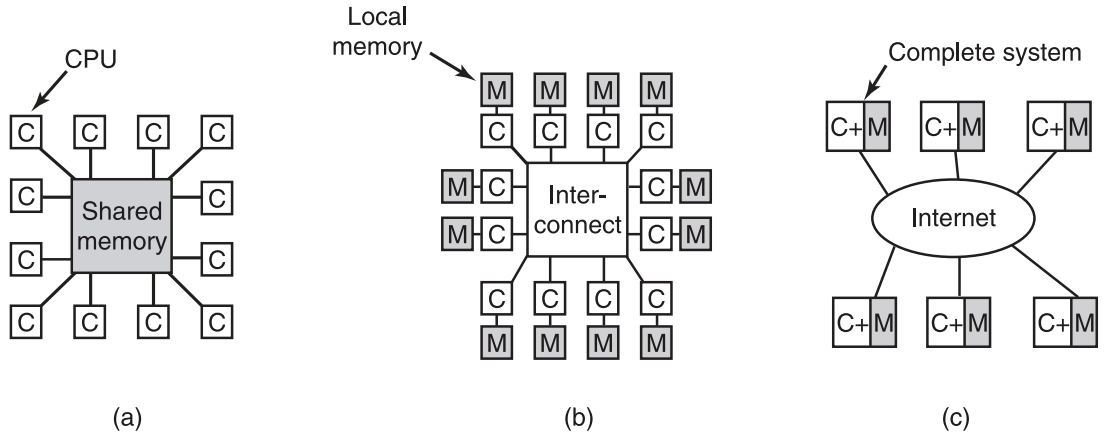


Figura 2 – (a) um multiprocessador de memória compartilhada. (b) um multicomputator com troca de mensagens. (c) um sistema distribuído de grande escala (TANENBAUM; BOS, 2014).

(v) Possuir sistemas memória distribuída restritivos, compostos por pequenas memórias locais; e
(vi) Apresentar componentes heterogêneos (*Compute Clusters* e *I/O Clusters*).

Alguns exemplos comerciais bem sucedidos de *lightweight manycores* são o Kalray MPPA-256 (DINECHIN et al., 2013), PULP (ROSSI et al., 2017) e Sunway SW26010 (FU et al., 2016). Especificamente, para o desenvolvimento deste trabalho será utilizado o processador Kalray MPPA-256. A Figura 3 apresenta uma visão geral do processador e suas peculiaridades, tais como:

(i) 288 núcleos de baixa frequência em um único *chip*;
(ii) núcleos organizados em 20 *clusters*;
(iii) 2 NoCs para transferência de dados entre *clusters*, uma para controle e outra para dados;
(iv) um sistema de memória distribuída composto por pequenas memórias locais, e.g., *Static Random Access Memory* (SRAM) de 2 MB;
(v) ausência de coerência de *cache*;
(vi) heterogeneidade: *clusters* destinados à computação (*Compute Clusters*) e *clusters* destinados à comunicação com periféricos (*I/O Clusters*).

2.2 NANVIX OS

O Nanvix[1] é um SO distribuído e de propósito geral que busca equilibrar desempenho, portabilidade e programabilidade para *lightweight manycores* (PENNA et al., 2019). O Nanvix é estruturado em três camadas de *kernel*. São elas:

[1] Disponível em https://github.com/nanvix

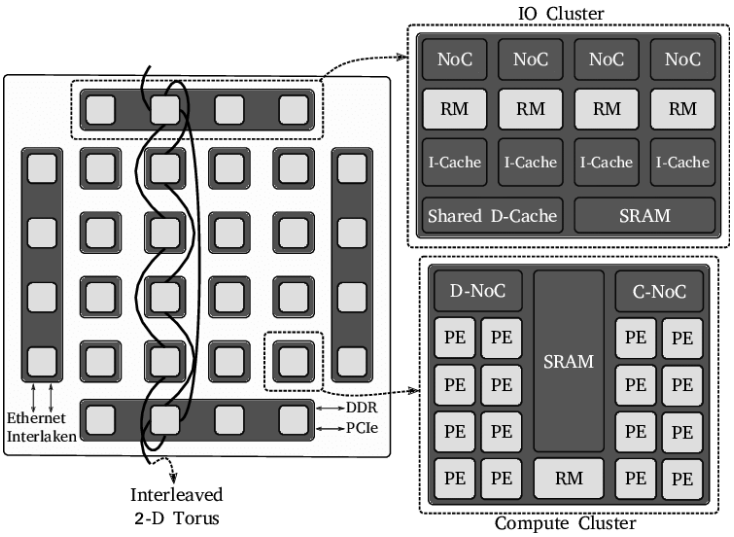


Figura 3 – Visão arquitetural do processador Kalray MPPA-256 (PENNA et al., 2019).

Nanvix *Hardware Abstraction Layer* (HAL) é a camada mais baixa que abstrai e provê o gerenciamento dos recursos de *hardware* sobre uma visão comum (PENNA; FRANCIS; SOUTO, 2019). Entre esses recursos estão: *cores*, *Translation Lookaside Buffer*s (TLBs), *cache*, *Memory Management Unit* (MMU), NoC, interrupções, memória virtual e recursos de *I/O*. De maneira geral, esta camada provê abstrações ao nivel do *core*, *cluster* e comunicação/sincronização entre *clusters* (PENNA, 2021). A Figura 4 ilustra a estrutura interna da HAL do Nanvix.

Nanvix *Microkernel* é a camada intermediária que provê gerenciamento de recursos e os serviços mínimos de um SO em um *cluster*. Entre esses serviços se encontram a comunição entre processos, gerenciamento de *threads* e memória, controle de acesso à memória e interface para chamadas de sistema. As chamadas de sistema podem ser executadas localmente, caso acessem dados *read-only* ou alterem estruturas internas do *core*, ou remotamente pelo *master core*, que atende à requisição e libera o *slave core* requisitante ao término da chamada (PENNA, 2021). Essa característica adjetiva o *microkernel* como assimétrico. A Figura 5 ilustra a estrutura interna do *microkernel* do Nanvix.

Nanvix *Multikernel* é a camada superior que provê os serviços mais complexos de um SO e dispõe uma visão a nível do processador em si. Os serviços são hospedados em *clusters*, i.e., isolados das aplicações de usuário. Os serviços atendem as requisições vindas dos processos de usuário através de um modelo cliente-servidor. As requisições e respostas são enviadas/recebidas através de passagem de mensagem via NoC. Os serviços dessa camada podem ser entendidos como fontes de informação que mantém a execução dos processos consistentes no processador, tendo em vista a natureza distribuída da memória nessas arquiteturas. Alguns serviços incluídos no Nanvix são mecanismos de *spawn* de processos e gerenciamento de nomes lógicos dos processos à fim a localização dos processos no processador.

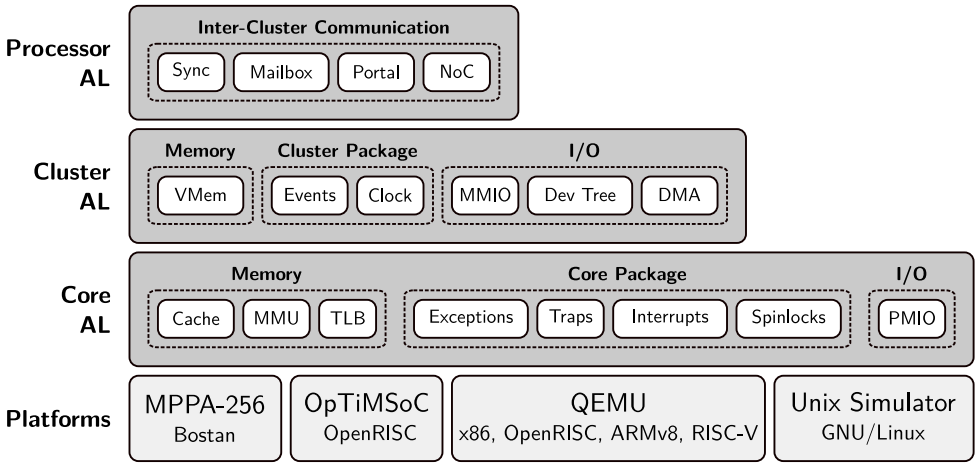


Figura 4 – Estrutura interna da HAL do Nanvix (PENNA, 2021).

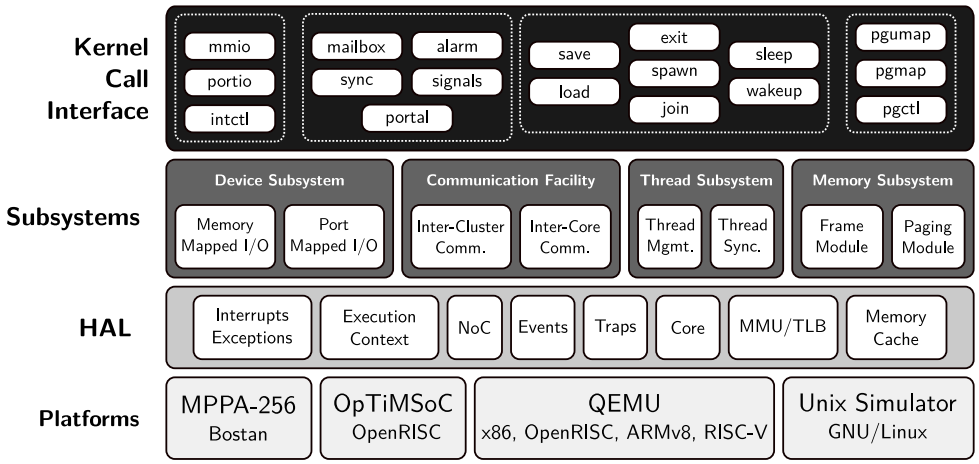


Figura 5 – Estrutura interna do *microkernel* do Nanvix (PENNA, 2021).

Em sua abordagem original, os processos no Nanvix são estáticos, i.e., cada *cluster* possui apenas um processo. Desse modo, uma vez que o processo inicia sua execução em um *cluster*, este finalizará a execução no mesmo *cluster*. Isso torna o processo dependente do *cluster* que o executa, fazendo com que a comunicação entre processos esteja atrelada aos *clusters* nos quais os processos são executados (e não aos processos em si). A falta de mobilidade dos processos nesse modelo pode trazer sobrecargas ao processador, afetando diretamente o o desempenho do sistema quando múltiplas aplicações estão em execução simultânea no processador. No caso de aplicações paralelas, compostas por múltiplos processos (ou *threads*) que se comunicam, a disposição dos processos (ou *threads*) nos *clusters* se torna importante, pois a comunicação entre *clusters* próximos é mais rápida e resulta em menor consumo energético do processador. Sendo assim, melhorar a mobilidade e a disposição dos processos no processador possibilitaria melhorar o gerenciamento dos recursos do mesmo. Um exemplo de mobilidade é viabilizar a migração de processos entre *clusters*. Neste contexto, este trabalho explora essa desassociação entre o processo e o *cluster* que o executa. Deste modo, nós aumentamos a mobilidade dos processos, podendo

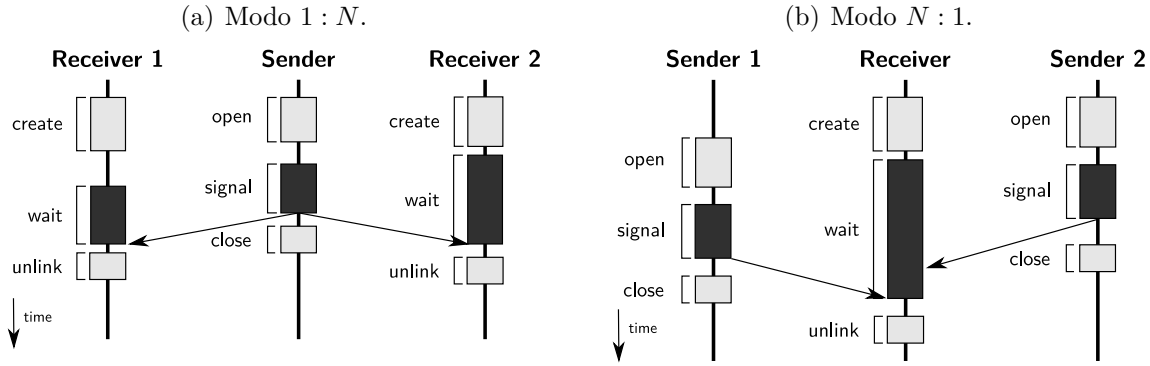


Figura 6 – Fluxo de execução da abstração *Sync* (PENNA, 2021).

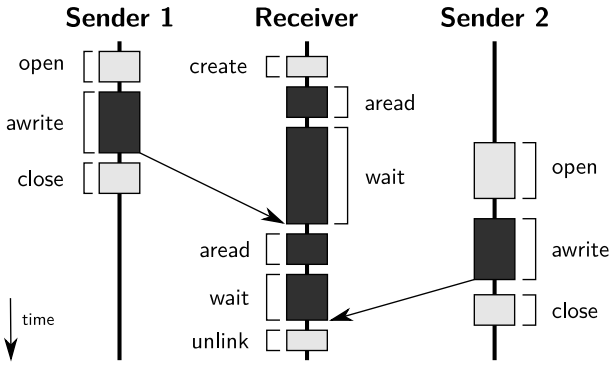


Figura 7 – Fluxo de execução da abstração *Mailbox* (PENNA, 2021)

permitir a migração de processos entre os *clusters* do processador.

2.2.1 Abstrações de Comunicação do Nanvix

O Nanvix dispõe de três abstrações de comunicações para transferência de dados e sincronização entre *clusters* (PENNA, 2021). Nas próximas seções serão detalhadas as três abstrações principais do Nanvix.

2.2.1.1 Sync

A abstração *Sync* suporta a sincronização entre *kernels*. Através dela um processo pode esperar um sinal, que pode ser disparado por outro processo remotamente através das interfaces NoC. Essa abstração é muito utilizada na inicialização do sistema para garantir um estado inicial consistente dos subsistemas do SO (PENNA, 2021).

O *Sync* pode ser operado duas maneiras distintas: o modo $1 : N$ e $N : 1$. No modo $1 : N$ (Figura 6(a)) um nó envia uma notificação a múltiplos nós, que estão esperando pelo sinal. Em contraste, no modo $N : 1$ (Figura 6(b)), múltiplos nós enviam uma notificação a um único nó (PENNA, 2021).

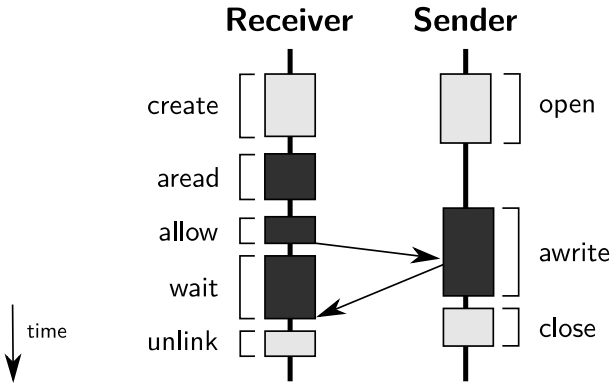


Figura 8 – Fluxo de execução da abstração *Portal* (PENNA, 2021)

2.2.1.2 Mailbox

A abstração *Mailbox* é responsável pelo suporte ao envio de mensagens de controle através da troca de pequenas mensagens de tamanho fixo. A abstração segue a semântica $N : 1$ e funciona da seguinte forma: um nó (destinatário da mensagem) possuí uma *Mailbox*, da qual lê mensagens, e múltiplos nós (remetentes da mensagem) podem escrever nessa *Mailbox* (PENNA, 2021). A Figura 7 ilustra o fluxo de execução da *Mailbox*.

2.2.1.3 Portal

A abstração portal suporta a troca de mensagens grandes e segue a semântica $1 : 1$. A abstração pode ter uso em diversos cenários que exigem grandes transferências de dados entre *clusters* (PENNA, 2021). A Figura 8 ilustra o fluxo de execução da abstração *Portal*.

2.3 VIRTUALIZAÇÃO E MIGRAÇÃO DE PROCESSOS

A virtualização é a desvinculação da execução de uma aplicação (ou até um SO) dos recursos físicos responsáveis pelo seu funcionamento. O desacoplamento entre aplicação e *hardware* pode permitir, dependendo da camada em que esta é aplicada, a existência simultânea e isolada de múltiplas instâncias de usuários ou SOs (Máquinas Virtuais (VMs)), que compartilham e concorrem pelos mesmos recursos de *hardware* reais. Reaproveitamento de recursos, portabilidade e segurança são algumas vantagens proporcionadas pela virtualização. Em ambientes *cloud* é muito comum a utilização de VMs para execução de tarefas nos servidores. Com o auxílio da virtualização, um único servidor pode alocar diversas VMs, possivelmente com SOs distintos.

O foco deste trabalho é a virtualização de processos. Isto é, o objetivo é desacoplar a execução de uma aplicação do *cluster* do *lightweight manycore* que a executa. Na abordagem original do Nanvix, o processo é dependente do *cluster* em que é alocado, o

que afeta o suporte a migração e diminui a eficiência computacional, como detalhado na Seção 2.2. Nesse contexto, a virtualização torna-se útil por aumentar a mobilidade dos processos, o que possibilitaria o gerenciamento da distribuição dos processos no processador. Especificamente, este trabalho explora um modelo mais leve de virtualização para *lightweight manycores* baseada em contêineres. Contêineres são executados pelo SO como aplicações virtuais e não incluem um SO convidado, resultando em um menor impacto no sistema de memória e requisitando menor complexidade do *hardware* (THALHEIM et al., 2018; SHARMA et al., 2016).

3 TRABALHOS RELACIONADOS

Neste capítulo serão mostradas técnicas e pesquisas que estão sendo desenvolvidas no que diz respeito à virtualização e migração de processos. Serão apresentados trabalhos relacionados, bem como serão evidenciadas as semelhanças e diferenças com o presente trabalho.

Grande parte das pesquisas relacionadas a migração estão inseridas em ambientes *cloud*. Nesses casos, os esforços estão voltados para redução do tempo total de migração, diminuição do *down time* (STOYANOV; KOLLINGBAUM, 2018; CLARK et al., 2005) e exploração/otimização das vantagens que a migração de processos oferece nesses ambientes computacionais. Entre elas podem-se citar:

(i) Balanceamento de carga (CHOUDHARY et al., 2017a; WANG et al., 2019);
(ii) Tolerância a falhas (FERNANDO et al., 2019);
(iii) Gerenciamento do consumo de energia (ALDOSSARY; DJEMAME, 2018);
(iv) Compartilhamento de recursos; e
(v) Manutenção de sistemas sem interrupções (CHOUDHARY et al., 2017a; WANG et al., 2019).

Os autores costumam aliar uma ou mais das características citadas acima para desenvolverem suas pesquisas. Por exemplo: Para evitar que muitos usuários tenham que fazer requisições à aplicações hospedadas em servidores distantes ou com muito tráfego, pode-se utilizar a transparência de localidade de execução de aplicações com a manutenção destas sem sua suspenção. Assim, é possível explorar algorítmos ou modelos para melhor posicionar as aplicações na rede com o objetivo de atender a maior quantidade de usuários, de maneira mais eficiente e sem que o sistema precise ser desligado (QIN et al., 2019). Ademais, esses modelos podem ser especializados para tipos específicos de aplicações, como *Internet of Things* (IoT) (WANG et al., 2019).

Além disso, pesquisas são feitas com o intuito de tornar a migração de VMs entre servidores tolerante a falhas (FERNANDO et al., 2019). A migração de VMs por algum motivo pode falhar (devido a problemas de rede, por exemplo). Nesse contexto, alguns autores sugerem o uso de *checkpoints*, os quais são estados consistentes da VM em dado momento. Se a migração falhar, a VM pode ser restaurada para um *checkpoint*. Isso evita que a VM seja perdida.

Em contraste com os trabalhos apresentados, nossa proposta não está inserida em ambientes *cloud*. Seu foco está na migração de processos entre *clusters* em um mesmo *chip*, em um sistema computacional restritivo que restringe as técnicas possíveis a serem utilizadas. Além disso, nosso trabalho se difere dos demais trabalhos por explorar a virtualização e migração de processos inserida no contexto dos *lightweight manycores* e não apenas buscando otimizar algoritmos específicos, e.g., *live migration*.

4 PROPOSTA DE VIRTUALIZAÇÃO E MIGRAÇÃO DE PROCESSOS PARA *LIGHTWEIGHT MANYCORES*

Este trabalho de conclusão propõe-se a aumentar a independência dos processos no processador através do projeto e desenvolvimento do suporte à virtualização e migração de processos em *lightweight manycores*. Ambientes *cloud*, nos quais o sistema de memória é de alta capacidade, usufruem da utilização de VMs para isolar duplicatas inteiras de SOs com o auxílio da virtualização a nível de instrução (SHARMA et al., 2016). Em oposição, *lightweight manycores* não dispõem de centenas de GBs de memória, mas sim pequenas memórias locais. Isso associado a outras simplificações de *hardware* faz com que algumas técnicas de virtualização sejam impraticáveis nesses ambientes computacionais.

Nesse contexto, visando atenuar o impacto da virtualização no sistema de memória, o presente trabalho explora um modelo de virtualização mais leve, baseado em contêineres adaptado para *lightweight manycores*. O SO executa os contêineres como aplicações virtuais. Sendo assim, não há a necessidade de um SO convidado, resultando em um menor impacto no sistema de memória e requisitando menor complexidade do *hardware* (THALHEIM et al., 2018; SHARMA et al., 2016).

4.1 SEPARAÇÃO KERNEL-USUÁRIO

4.1.1 Divisão de Dados e Instruções

Para a virtualização de processos através da conteinerização, é recomendável que as informações relevantes para a manipulação dos processos em execução estejam isoladas das informações internas do próprio SO para que os recursos de *hardware* sejam utilizados de maneira eficiente (CHOUDHARY et al., 2017b). A Figura 9(a) ilustra como os subsistemas do Nanvix são originalmente estruturados. Não há uma divisão explícita do que são dados para funcionamento interno do SO ou dependências locais do processo. Esta abordagem torna algumas das funcionalidades do SO onerosas porque ela dificulta o acesso às informações do processo e impacta partes independentes do sistema, e.g., migração e segurança dos processos.

Além disso, a geração original de um executável do Nanvix compila todos os níveis em bibliotecas estáticas (HAL, *microkernel*, *libnanvix*, *ulibc* e *multikernel*) e as junta com a aplicação do usuário de forma a misturar o que é *kernel* do que é usuário. Visando a separação das informações entre usuário e *kernel*, nós adaptamos o *script* de ligação original do Nanvix. Na nova versão, as seções .text, .data, .bss e .rodata dos arquivos binários compilados são renomeados, especificando qual camada de abstração tal arquivo pertence. Desta forma, é possível identificar dados e instruções de cada camada do Nanvix, assim como as informações do usuário. Sendo assim, são geradas seções .text, .data, .rodata e .bss específicas para o *kernel* e usuário. Portanto, todas as informações de

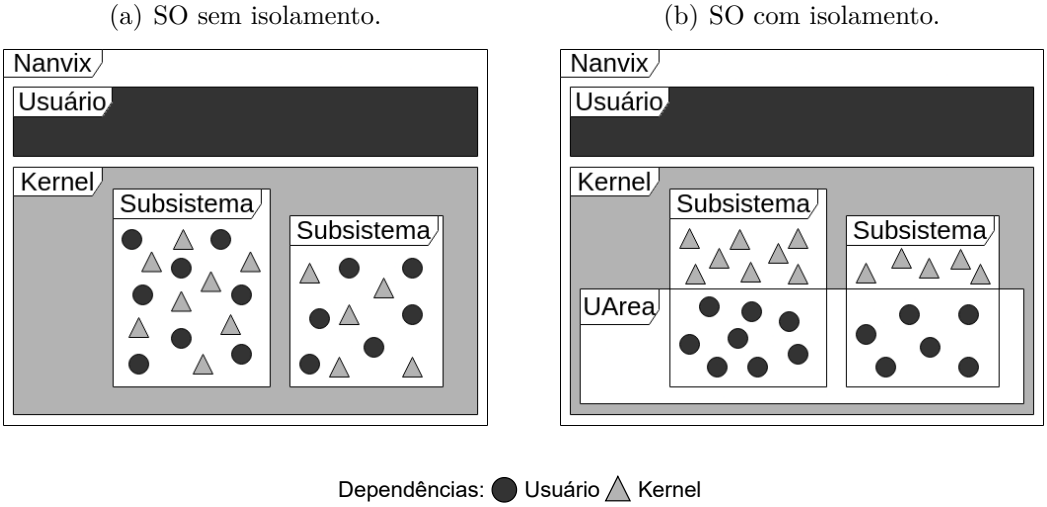


Figura 9 – Diferença da estrutura do Nanvix com e sem a *User Area*.

kernel, alocadas nos endereços mais baixos da memória, são isoladas das informações de aplicação, alocadas nos endereços mais altos da memória. Neste processo, são exportadas algumas constantes que apontam onde começam e terminam as partes do binário que são relacionadas ao *kernel* e à aplicação. Essas constantes permitem a manipulação e gerenciamento mais precisos dos segmentos de memória do *kernel* e da aplicação.

Através dessa estratégia, todos os *clusters* passam a ter a mesma organização interna de *kernel*, facilitando a migração. Ou seja, a migração pode ser feita através do *salvamento* dos dados e instruções da aplicação de um *cluster* e *restauração* dos mesmos nas respectivas posições em outro *cluster*. Essas posições são identificadas pelas constantes exportadas no processo de compilação. Com isso, evita-se manipulações mais complexas do processo como a busca em várias regiões de memória para montar o estado interno do processo.

4.2 *USER AREA*

Além da separação de dados e instruções entre *kernel* e aplicação, é necessário a identificação e separação das estruturas internas do SO que são manipuladas pelo usuário e constituem o estado interno do processo. Nesse contexto, é introduzido o conceito de conteinerização para isolar as dependências que o usuário possui dentro do *cluster*. Ou seja, nós isolamos os dados que são gerenciados pelo *kernel* mas pertencem ao contexto do processo de usuário. Neste contexto, nós isolamos tais dados em uma região de memória bem definida, denominada de *User Area* (UArea).

Detalhadamente, a UArea mantém informações sobre:

(i) *threads* ativas, incluindo identificadores e contextos;
(ii) ponteiros para suas pilhas de execução;
(iii) variáveis de controle e filas de escalonamento;

(iv) estruturas de gerenciamento de chamadas de sistema; e
(v) estruturas de gerenciamento de memória (e.g., sistema de paginação).

Essa estrutura genérica foi projetada para englobar as várias arquiteturas suportadas pelo Nanvix. Além disso, a estrutura permite a modificação e expansão, não se limitando ao estado atual do desenvolvimento do Nanvix, para atender os objetivos de outros projetos que usufruam do Nanvix.

4.3 MIGRAÇÃO DE PROCESSOS

Como aplicação direta do isolamento do processo, a migração de processos torna-se viável. Especificamente, nós eliminamos a necessidade de descobrir quais são e onde estão as informações que compõem o estado de um processo dentro do Nanvix através da criação de uma instância isolada do espaço do usuário via conteinerização, facilitando a transferência de seu contexto. Isso só é possível porque os *clusters* possuem uma estrutura de *kernel* idêntica (devido às mudanças desenvolvidas no processo de compilação detalhados na Subseção 4.1.1). Por este motivo, eliminamos o envio de dados redundantes entre *clusters* referentes à instância local do SO, atenuando o impacto da migração sobre a NoC.

4.3.1 Rotina de migração

Para a migração de um processo entre *clusters* foi desenvolvida uma rotina de migração. A funcionalidade é similar ao *Checkpoint/Restore In Userspace* (CRIU), ferramenta utilizada por *softwares* de gerenciamento de contêineres como o Docker. Porém, a migração será executada por intermédio de *daemons* do SO. Nesta versão inicial do projeto, implementaremos o algoritmo *hot migration* para migração de processos. A técnica de *hot migration* migra a aplicação durante sua execução, copiando as páginas de memória e o estado de execução da aplicação. A seguir é detalhado o fluxo de execução da migração:

1. **Congelamento da execução do processo em um estado consistente.**
   Antes do envio do processo a outro *cluster*, é necessário que este esteja em um estado consistente e estático. Isso significa que durante o processo de migração é preciso que todas as operações dele sejam pausadas. Isso com o intuito de evitar inconsistências que podem ser causadas por condições de corrida. Para atingir esse estado consistente, a chamada de sistema *freeze* é invocada. Esta é uma chamada de sistema que é tratada apenas pelo *master core*. Especificamente, esta chamada impede o escalonamento de *threads* de aplicação, i.e., *threads* que não executam no *master core*. Isso garante uma pausa na aplicação sem que o SO seja impedido de executar, o que é imprescindível para a migração, já que as informações do

processo precisam ser enviadas pelas interfaces NoC do *cluster* remetente, o que exige que o SO atenda às requisições de envio de dados. Após o congelamento da aplicação, são verificados os *buffers* de chamadas de sistema. Após o travamento no escalonamento de *threads* de usuário, novas chamadas de sistema requisitadas pela aplicação não podem ocorrer. Contudo, a fim de evitar a perda de qualquer chamada que possa ter sido requisitada antes do congelamento do escalonamento, os *buffers* de chamada de sistema são verificados. Todas as chamadas de sistema vindas de *threads* de aplicação que são encontradas, são tradadas antes da migração. Após o congelamento do escalonamento e verificação nos *buffers* de chamada de sistema, o processo é considerado consistente e seu contexto pode ser migrado.

2. **Transferência do contexto do processo entre *clusters*.**
   Com o processo em um estado consistente, uma *task* de sistema, que é executada no *master core*, é criada para o envio dos dados ao *cluster* destinatário. Através das abstrações de comunicação *Mailbox* e *Portal*, as seções de dados e instruções do processo são enviadas ao *cluster* destinatário. Logo após, a UArea é enviada. O envio de dados, instruções e UArea garantem que o contexto inteiro do processo seja enviado, possibilitando a retomada da execução no *cluster* destinatário.
3. **Restauração da execução do processo no *cluster* destino.**
   Com o contexto do processo já no *cluster* destinatário, a execução é restaurada. Isso é feito pela chamada de sistema *unfreeze*, que descongela o escalonamento de *threads* de usuário. Assim, a execução do processo continua normalmente, agora em outro *cluster*.

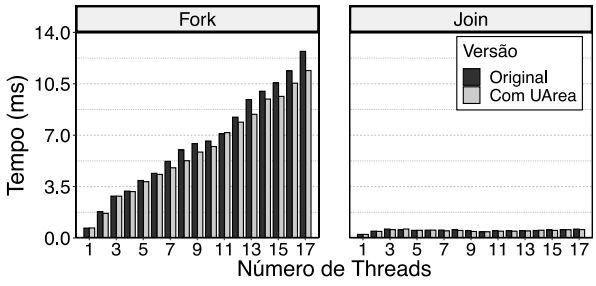
5 RESULTADOS PARCIAIS

A solução foi avaliada em etapas anteriores ao desenvolvimento atual do trabalho e os resultados seguintes englobam apenas o susbsistema de *threads* do Nanvix. Para avaliar o impacto das mudanças feitas para a virtualização, foram desenvolvidos experimentos sobre a manipulação de *threads* e suporte à migração de processos no Nanvix. Todos os experimentos foram executados no processador Kalray MPPA-256 e os resultados mostrados são valores médios de 100 replicações de cada experimento para garantir 95% de confiança estatística, resultando em um desvio padrão inferior a 1%.

O experimento de manipulação de *threads* mensura os impactos na criação e junção através de diferentes perspectivas. Especificamente, coletamos o tempo de execução, desvios e faltas ocorridas na *cache* de dados e de instrução (Figura 10). Os resultados apresentam um aumento no desempenho das operações de manipulação quando utilizamos a UArea porque exploramos melhor a localidade espacial dos dados, o que, consequentemente, diminui o número de faltas na *cache*.

O experimento de migração avaliou o tempo de transferência de um processo entre *clusters*. A aplicação de usuário migrada contém 352,8 KB. Detalhadamente, foram transferidos instruções e dados (342,8 KB), a UArea (2 KB) e uma pilha de execução (8 KB). O *down time* médio da aplicação, i.e., o tempo que a aplicação demorou para restaurar a execução no *cluster* destinatário após a migração, foi de 226 ms. A média de tempo para o *cluster* remetente enviar todos os dados foi de 218 ms.

(a) Tempo de criação de *threads*.



(b) Métricas do *kernel*.

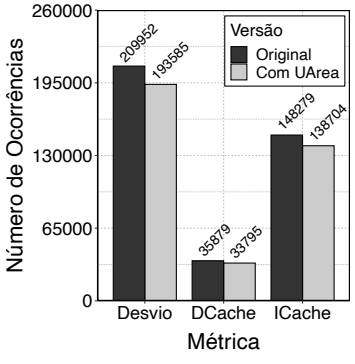


Figura 10 – Impactos da virtualização sobre a manipulação de *threads*.

6 CONCLUSÕES

Neste trabalho, foi explorado um modelo de virtualização leve baseada em contêineres que considera as restrições arquiteturais dos *lightweight manycores*, adaptando-se as suas restrições, principalmente relacionadas à memória. A virtualização proposta visa melhorar a mobilidade e gerenciamento de processos para *lightweight manycores* no contexto de em um Sistema Operational (SO) distribuído. Os resultados mostraram que o isolamento das dependências de um processo aumentaram o desempenho de operações do *kernel* e suportaram de fato a migração de processos de forma eficiente. Como trabalhos futuros, pretende-se:

(i) Ampliar a virtualização, englobando outros subsistemas do Nanvix;
(ii) Habilitar a execução simultânea de múltiplas aplicações no processador e sua proteção;
(iii) Realizar uma maior quantidade de experimentos para avaliar os sobrecustos introduzidos pela abordagem proposta.

REFERÊNCIAS

ALDOSSARY, M.; DJEMAME, K. Performance and energy-based cost prediction of virtual machines live migration in clouds. In: **CLOSER**. [S.l.: s.n.], 2018. p. 384–391.

ASMUSSEN, N. et al. M3: A hardware/operating-system co-design to tame heterogeneous manycores. In: **ASPLOS '16 Proceedings of the Twenty-First International Conference on Architectural Support for Programming Languages and Operating Systems**. ACM. (ASPLOS '16, v. 44), p. 189–203. ISBN 978-1-4503-4091-5. Disponível em: http://dl.acm.org/citation.cfm?doid=2980024.2872371.

CASTRO, M. et al. Seismic wave propagation simulations on low-power and performance-centric manycores. **Parallel Computing**, v. 54, p. 108–120, 2016. ISSN 01678191. Disponível em: http://www.sciencedirect.com/science/article/pii/S0167819116000417.

CHOUDHARY, A. et al. A critical survey of live virtual machine migration techniques. **Journal of Cloud Computing**, SpringerOpen, v. 6, n. 1, p. 1–41, 2017.

CHOUDHARY, A. et al. A critical survey of live virtual machine migration techniques. **Journal of Cloud Computing**, SpringerOpen, v. 6, n. 1, p. 1–41, 2017.

CLARK, C. et al. Live migration of virtual machines. In: **Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2**. [S.l.: s.n.], 2005. p. 273–286.

DINECHIN, B. D. de et al. A clustered manycore processor architecture for embedded and accelerated applications. In: **2013 IEEE High Performance Extreme Computing Conference (HPEC)**. [S.l.: s.n.], 2013. p. 1–6.

FERNANDO, D. et al. Live migration ate my vm: Recovering a virtual machine after failure of post-copy live migration. In: IEEE. **IEEE INFOCOM 2019-IEEE Conference on Computer Communications**. [S.l.], 2019. p. 343–351.

FRANCESQUINI, E. et al. On the Energy Efficiency and Performance of Irregular Application Executions on Multicore, NUMA and Manycore Platforms. **Journal of Parallel and Distributed Computing (JPDC)**, v. 76, n. C, p. 32–48, fev. 2015. ISSN 0743-7315. Disponível em: http://linkinghub.elsevier.com/retrieve/pii/S0743731514002093.

FU, H. et al. The sunway taihulight supercomputer: system and applications. **Science China Information Sciences**, Springer, v. 59, n. 7, p. 1–16, 2016.

KELLY, B.; GARDNER, W.; KYO, S. AutoPilot: Message Passing Parallel Programming for a Cache Incoherent Embedded Manycore Processor. In: **Proceedings of the 1st International Workshop on Many-core Embedded Systems**. Tel-Aviv, Israel: ACM, 2013. (MES '13), p. 62–65. ISBN 978-1-4503-2063-4. Disponível em: http://dl.acm.org/citation.cfm?doid=2489068.2491624.

KLUGE, F.; GERDES, M.; UNGERER, T. An operating system for safety-critical applications on manycore processors. In: **2014 IEEE 17th International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing**. IEEE. (ISORC '14), p. 238–245. ISBN 978-1-4799-4430-9. Disponível em: http://ieeexplore.ieee.org/document/6899155/.

KOGGE, P. et al. Exascale computing study: Technology challenges in achieving exascale systems. **Defense Advanced Research Projects Agency Information Processing Techniques Office (DARPA IPTO), Techinal Representative**, v. 15, 01 2008.

MOORE, G. E. Cramming more components onto integrated circuits. **Electronics**, v. 38, n. 8, April 1965.

PENNA, P. H. **Nanvix: A Distributed Operating System for Lightweight Manycore Processors**. Tese (Doutorado) — Université Grenoble Alpes, 2021.

PENNA, P. H. et al. Using the Nanvix Operating System in Undergraduate Operating System Courses. In: **2017 VII Brazilian Symposium on Computing Systems Engineering**. Curitiba, Brazil: IEEE, 2017. (SBESC '17), p. 193–198. ISBN 978-1-5386-3590-2. Disponível em: http://ieeexplore.ieee.org/document/8116579/.

PENNA, P. H.; FRANCIS, D.; SOUTO, J. The Hardware Abstraction Layer of Nanvix for the Kalray MPPA-256 Lightweight Manycore Processor. In: **Conférence d'Informatique en Parallélisme, Architecture et Système**. Anglet, France: [s.n.], 2019. Disponível em: https://hal.archives-ouvertes.fr/hal-02151274.

PENNA, P. H. et al. Using The Nanvix Operating System in Undergraduate Operating System Courses. In: **VII Brazilian Symposium on Computing Systems Engineering**. Curitiba, Brazil: [s.n.], 2017. Disponível em: https://hal.archives-ouvertes.fr/hal-01635880.

PENNA, P. H. et al. On the Performance and Isolation of Asymmetric Microkernel Design for Lightweight Manycores. In: **SBESC 2019 - IX Brazilian Symposium on Computing Systems Engineering**. Natal, Brazil: [s.n.], 2019.

PENNA, P. H. et al. Inter-kernel communication facility of a distributed operating system for noc-based lightweight manycores. **Journal of Parallel and Distributed Computing**, Elsevier, v. 154, p. 1–15, 2021.

PENNA, P. H. et al. RMem: An OS Service for Transparent Remote Memory Access in Lightweight Manycores. In: **MultiProg 2019 - 25th International Workshop on Programmability and Architectures for Heterogeneous Multicores**. Valencia, Spain: [s.n.], 2019. (High-Performance and Embedded Architectures and Compilers Workshops (HiPEAC Workshops)), p. 1–16. Disponível em: https://hal.archives-ouvertes.fr/hal-01986366.

QIN, J. et al. Online user distribution-aware virtual machine re-deployment and live migration in sdn-based data centers. **IEEE Access**, v. 7, p. 11152–11164, 2019.

ROSSI, D. et al. Energy-efficient near-threshold parallel computing: The pulpv2 cluster. **IEEE Micro**, v. 37, n. 5, p. 20–31, 2017.

SHARMA, P. et al. Containers and virtual machines at scale: A comparative study. In: **Proceedings of the 17th International Middleware Conference**. [S.l.: s.n.], 2016. p. 1–13.

STOYANOV, R.; KOLLINGBAUM, M. J. Efficient live migration of linux containers. In: SPRINGER. **International Conference on High Performance Computing**. [S.l.], 2018. p. 184–193.

TANENBAUM, A. S.; BOS, H. **Modern Operating Systems**. 4th. ed. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014. ISBN 013359162X, 9780133591620.

THALHEIM, J. et al. Cntr: Lightweight os containers. In: **2018 USENIX Annual Technical Conference**. [S.l.: s.n.], 2018. p. 199–212.

VANZ, N.; SOUTO, J. V.; CASTRO, M. Virtualização e migração de processos em um sistema operacional distribuído para lightweight manycores. In: SBC. **Anais da XXII Escola Regional de Alto Desempenho da Região Sul**. [S.l.], 2022. p. 45–48.

WANG, Z. et al. Ada-things: An adaptive virtual machine monitoring and migration strategy for internet of things applications. **Journal of Parallel and Distributed Computing**, Elsevier, v. 132, p. 164–176, 2019.