

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
29 de junho de 2022

Nicolas Vanz

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

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de Bacharel em Ciência da Computação e aprovado em sua forma final pelo curso de Graduação em Ciência da Computação.

Florianópolis, 29 de junho de 2022.

Prof. Renato Cislighi, Dr.
Coordenador do Curso

Banca Examinadora:

Prof. Márcio Bastos Castro, Dr.
Orientador
Universidade Federal de Santa Catarina

Nicolas Vanz

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

Trabalho de Conclusão de Curso submetido ao Curso de Graduação em Ciência da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Márcio Bastos Castro, Dr.

Coorientador: Prof. João Vicente Souto, Dr.

Florianópolis

29 de junho de 2022

m

Dedico este trabalho aos meus pais,
aos demais membros da família e aos meus amigos.

AGRADECIMENTOS

Agradeço a Deus pela minha vida.

Agradeço aos meus pais e minha família, que sempre se me motivaram e confiaram na minha capacidade de superar os obstáculos da vida.

Agradeço a todos que de alguma forma contribuíram no desenvolvimento deste trabalho e me auxiliaram na jornada de me tornar um profissional mais capaz. Em especial, agradeço ao meu professor orientador Márcio Bastos Castro, ao meu coorientador João Vicente Souto e aos demais colegas de projeto.

Agradeço ao Conselho Nacional de Desenvolvimento Científico e Tecnológico pelo auxílio através do Programa Institucional de Bolsas de Iniciação Científica (PIBIC).

Agradeço aos meus amigos de curso pela convivência intensa e companheirismo durante os últimos anos.

So we keep asking, over and over, until a handful
of earth stops our mouths — but is that an answer?
(Heine, H., *The Lazarus Poems*, 1851)

RESUMO

A classe de processadores *lightweight manycores* 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 Operacional (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 em desenvolvimento e de código aberto 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. containerização

LISTA DE FIGURAS

Figura 1 – Visão conceitual de um processador <i>lightweight manycore</i> (PENNA et al., 2021)	20
--	----

LISTA DE ALGORITMOS

LISTA DE SÍMBOLOS

\leftarrow	Atribuição
\exists	Quantificação existencial
\rightarrow	Implicação
\wedge	E lógico
\vee	Ou lógico
\neg	Negação lógica
\mapsto	Mapeia para
\sqsubseteq	Subclasse (em ontologias)
\subseteq	Subconjunto: $\forall x . x \in A \rightarrow x \in B$
$\langle \dots \rangle$	Tupla
\forall	Quantificação universal
mmmmm	Nenhum sentido, apenas estou aqui para demonstrar a largura máxima dessas colunas. Ao abrir o ambiente <code>listadesimbolos</code> , pode-se fornecer um argumento opcional indicando a largura da coluna da esquerda (o default é de 5em): <code>\begin{listadesimbolos}[2cm] \end{listadesimbolos}</code>

SUMÁRIO

1	INTRODUÇÃO	19
1.1	OBJETIVOS	21
1.1.1	Objetivo Principal	21
1.1.2	Objetivos Específicos	21
1.2	ORGANIZAÇÃO DO TRABALHO	21
2	REFERENCIAL TEÓRICO	23
2.1	<i>LIGHTWEIGHT MANYCORES</i>	23
2.2	NANVIX	23
2.3	VIRTUALIZAÇÃO E CONTEINERIZAÇÃO	24
	REFERÊNCIAS	25
	APÊNDICE A – ARTIGO CIENTÍFICO	27

1 INTRODUÇÃO

Durante anos o aumento do desempenho em processadores esteve associada ao aumento da frequência interna nos processadores e avanços na tecnologia dos semicondutores. Essas técnicas se mantiveram eficientes até o momento em que a dissipação de calor interna dos *chips* inviabilizou o aumento da frequência dos processadores. Isso, associado ao fim da lei de Moore (MOORE, 1965) fez com que novas maneiras de se aumentar o poder computacional fossem exploradas.

Como alternativa para o aumento de desempenho, foram desenvolvidos os processadores com vários núcleos de processamento, os *multicores*, cujo desempenho vem aliado também à quantidade de núcleos, e não mais apenas às altas frequências de relógio. Desse modo, mesmo com a estabilização da frequência 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 ampliar.

Atualmente, a eficiência energética dos sistemas computacionais revela-se tão importante quanto o desempenho. Segundo o Departamento de Defesa do Governo dos Estados Unidos (DARPA/IPTO), 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 (KOGGE et al., 2008). Nesse cenário surge a classe dos processadores *lightweight manycores*. Esses processadores são classificados como Multiprocessor System-on-Chips (MPSoCs) e tem como objetivo justamente atrelar alto desempenho à eficiência energética (FRANCESQUINI et al., 2015). Para atingir esse objetivo, a arquitetura dessa classe de processadores é caracterizada por:

- (i) Integrar centenas ou milhares de núcleos de processamento operando a baixas frequências em um único chip;
- (ii) Operar sobre *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 de memória distribuídos e restritivos; e
- (vi) Apresentar componentes heterogêneos.

Os 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. Uma visão conceitual da arquitetura de um *lightweight manycore* é ilustrada pela Figura 1.

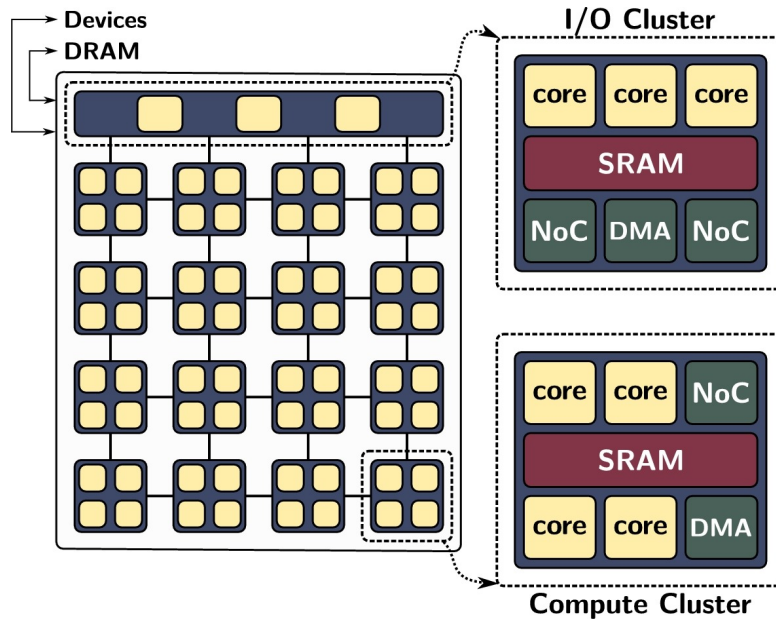


Figura 1 – Visão conceitual de um processador *lightweight manycore* (PENNA et al., 2021)

Apesar de os processadores *lightweight manycores* serem uma alternativa às abordagens tradicionais no que se refere ao aumento de desempenho, as características arquiteturais ainda induzem problemas de programabilidade nas aplicações paralelas (CASTRO et al., 2016). Entre eles podem-se citar:

- (i) Modelo de programação híbrida que força troca de informação entre os *clusters* exclusivamente por troca de mensagens via NoC (KELLY; GARDNER; KYO, 2013);
- (ii) Sistema de memória restritivo, em que há múltiplos espaços de endereçamento, pequena memória local, necessidade de busca em memória remota e separação da memória em pequenos blocos explicitamente para a manipulação dos dados (CASTRO et al., 2016);
- (iii) Latência e gargalos de comunicação na NoC;
- (iv) Falta de suporte de coerência de cache para economia de energia, o que exige do programador a gerência de cache via *software*;
- (v) Configuração heterogênea no que se refere aos *Compute Clusters* e *I/O Clusters*, o que dificulta o desenvolvimento de aplicações;

Atualmente, alguns estudos são feitos para amenizar o impacto da arquitetura sobre o desenvolvimento de aplicações. Neles, sobressaem-se os SOs distribuídos, que garantem um ambiente mais robusto e rico (ASMUSSEN et al., ; KLUGE; GERDES; UNGERER, ; PENNA et al., 2019). Destaca-se ainda os estudos em SOs distribuídos baseados em uma abordagem *multikernel* (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 para maior eficiência do mesmo (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 para isolar duplicatas inteiras de SOs com a ajuda de virtualização no 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 manycore* 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 requerendo 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 adaptar o Nanvix, um para *lightweight manycores*, de modo que os recursos utilizados por um processo sejam virtualizados. Isso com o objetivo de desvincular a execução de um processo com o local i.e., *cluster* onde está alocado e aumentar a mobilidade de processos no processador.

1.1.2 Objetivos Específicos

- (i) Propor um modelo de virtualização adaptado às necessidades e imposições de um *lightweight manycore*;
- (ii) Implementar o modelo proposto no Nanvix, um 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 na execução normal do Nanvix;

1.2 ORGANIZAÇÃO DO TRABALHO

As próximas seções do trabalho estão organizadas da seguinte maneira. No Capítulo 2 serão apresentados alguns conceitos importantes para o melhor entendimento do trabalho. Dentre esses conceitos pode-se citar: (i) *Lightweight manycores*; (ii) Multi-

processadores; (iii) Multicomputadores; (iv) Virtualização. Além disso, será detalhado o Sistema Operacional (SO) e o *lightweight manycore* que será utilizado neste trabalho.

2 REFERENCIAL TEÓRICO

incluir base multiprocessadores/multicomputadores nesta seção

2.1 *LIGHTWEIGHT MANYCORES*

2.2 NANVIX

overview/threads/comunicação

O Nanvix¹ é um Sistema Operacional (SO) distribuído e de propósito geral que busca equilibrar desempenho, portabilidade e programabilidade para *lightweight many-cores* (PENNA et al., 2019). Nanvix é estruturado em 3 camadas de *kernel*. São elas:

adicionar figura da hal, microkernel e multikernel do nanvix

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 eles estão: *cores*, *Translation Lookaside Buffers* (TLBs), *cache*, *Memory Management Unit* (MMU), *Network-on-Chip* (NoC), interrupções, memória virtual, recursos de *I/O*. De maneira geral, esta camada provê visões a nível de *core*, *cluster* e comunicação/sincronização entre *clusters* (PENNA, 2021)

Nanvix *Assymetric 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 comunicação intercluster, 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 ou alterem estruturas internas do *core*, ou remotamente pelo *master core* que atende à requisição e libera o *slave core* requisitante ao seu término (PENNA, 2021). Essa característica adjetiva o *microkernel* como assimétrico.

Nanvix *Multikernel* é a camada superior que provê os serviços 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 e 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. Neles estão presentes mecanismos de

¹ Disponível em <https://github.com/nanvix>

de processos e obtenção de nomes lógicos dos processos (a fim de localizá-los para comunicação), por exemplo.

continuar daqui

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 e.g., a comunicação entre processos está 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 e afeta o suporte a multi-aplicação. Por exemplo, 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 i.e., viabilizar a migração de processos entre *clusters*, possibilitaria melhorar o gerenciamento dos recursos do mesmo.

2.3 VIRTUALIZAÇÃO E CONTEINERIZAÇÃO

REFERÊNCIAS

- 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>.
- 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.
- 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), Techninal 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>.

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.

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.

APÊNDICE A – ARTIGO CIENTÍFICO

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

Nicolas Vanz¹, João Vicente Souto¹, Márcio Castro¹

¹Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD)
Universidade Federal de Santa Catarina (UFSC) - Florianópolis/SC

nicolas.vanz@grad.ufsc.br, joao.vicente.souto@posgrad.ufsc.br,
marcio.castro@ufsc.br

Resumo. *Lightweight manycores surgiram para prover um alto grau de paralelismo e eficiência energética. Contudo, suas peculiaridades arquitetônicas introduzem dificuldades significativas no gerenciamento de recursos. Especialmente, o gerenciamento de processos precisa mitigar problemas provenientes das pequenas memórias locais e da falta de um suporte robusto para virtualização. Neste contexto, o presente trabalho explora um método de virtualização mais leve baseado em contêineres para um sistema operacional distribuído. Os resultados mostram que o método proposto provê uma melhor localidade dos dados e possibilita efetivamente a migração de processos.*

1. Introdução

Atualmente, a eficiência energética de sistemas computacionais revela-se tão importante quanto seu desempenho. Processadores *lightweight manycores* surgem para aliar alto desempenho à eficiência energética. Contudo, as escolhas arquiteturais necessárias para atingir esse objetivo dificultam o desenvolvimento de aplicações para essa classe de processadores [Castro et al. 2016].

O Kalray MPPA-256 é um exemplo comercial de *lightweight manycore* e exemplifica as características dessa classe de processadores. A Figura 1 apresenta uma visão geral do Kalray MPPA-256 e suas peculiaridades, tais como: (i) integrar 288 de núcleos de baixa frequência em um único chip; (ii) organizar os núcleos em 20 conjuntos (*clusters*) para compartilhamento de recursos locais; (iii) utilizar 2 *Network-on-Chips* (NoCs) para transferência de dados entre *clusters*; (iv) possuir um sistema de memória distribuída composto por pequenas memórias locais, e.g., 2 MB; (v) não dispor de coerência de *cache*; e (vi) apresentar componentes heterogêneos, e.g., *clusters* destinados à computação ou comunicação com periféricos.

Tais características, especialmente relacionadas à memória, inviabilizam um suporte complexo para virtualização. Por exemplo, máquinas virtuais utilizadas em ambientes *cloud* possuem à disposição centenas de GBs para isolar duplicatas inteiras de Sistemas Operacionais (SOs) com a ajuda de virtualização no nível de instrução [Sharma et al. 2016]. As pequenas memórias locais e a simplificação do *hardware* para reduzir o consumo energético restringe os tipos de virtualização suportados. Neste contexto, o presente trabalho explora um modelo de virtualização para *lightweight manycore* baseado em contêineres. Contêineres são executados pelo SO como aplicações virtuais e não incluem um SO hospedeiro, resultando em um menor impacto no sistema de memória e requerendo menor complexidade do *hardware* [Thalheim et al. 2018, Sharma et al. 2016].

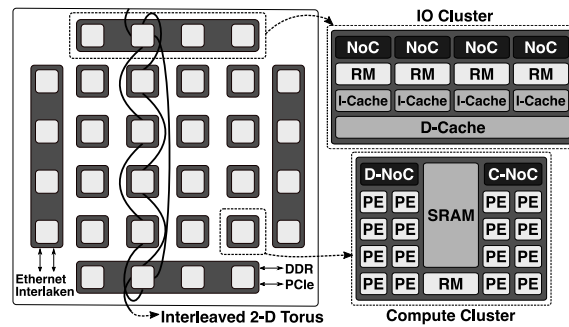


Figura 1. Visão arquitetural do processador Kalray MPPA-256 [Penna et al. 2019].

2. Sistema Operacional Nanvix

O Nanvix¹ é um SO distribuído e de propósito geral que busca equilibrar desempenho, portabilidade e programabilidade para *lightweight manycores* [Penna et al. 2019]. Nanvix é estruturado em 3 camadas de *kernel*. São elas: (i) Nanvix Hardware Abstraction Layer (HAL): é a camada mais baixa que abstrai os recursos de *hardware* sobre uma visão comum. (ii) Nanvix Microkernel: é a camada intermediária que provê gerenciamento de recursos e os serviços mínimos de um SO em um *cluster*. (iii) Nanvix Multikernel: é a camada superior que provê os serviços de um SO. Os serviços atendem as requisições vindas dos processos de usuário através de um modelo cliente-servidor.

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*. A falta de mobilidade dos processos nesse modelo pode trazer sobrecargas ao processador e afetam o suporte a multi-aplicação. Por exemplo, 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.

3. Virtualização e Migração de Processos

Visando facilitar o desenvolvimento de aplicações para essa classe de processadores e tornar o gerenciamento de recursos mais transparente para o programador, este trabalho explora um modelo de virtualização mais leve baseado em contêineres.

Neste contexto, 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. 2017]. A Figura 2a ilustra como os subsistemas do Nanvix são 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.

Para tornar a manipulação de processos mais eficiente, introduzimos conceitos de containerização, isolando as dependências que o usuário possui dentro do *cluster* (dados que são gerenciados pelo *kernel* mas pertencem ao contexto do processo de usuário). A distinção entre usuário e SO ocorre pela separação das instruções e dados de cada um

¹Disponível em <https://github.com/nanvix>

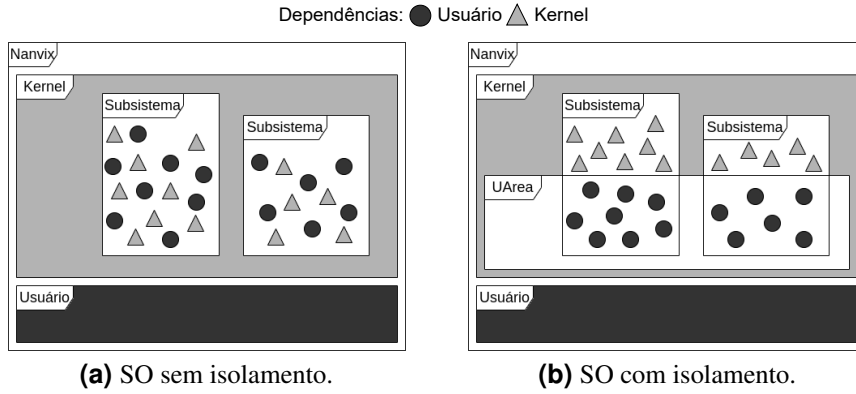


Figura 2. Diferença da estrutura do Nanvix com e sem a *User Area*.

em segmentos de memória diferentes. Além disso, isolamos as dependências internas do processo em uma área de memória bem definida, separada das demais estruturas internas do *kernel*, denominada *User Area (UArea)*. A Figura 2b ilustra conceitualmente a divisão das dependências fornecida pela UArea.

Especificamente, a UArea mantém informações sobre (i) as *threads* ativas, i.e., seus identificadores e respectivos contextos; (ii) variáveis de controle e filas de escalonamento; (iii) ponteiros para suas pilhas de execução; e (iv) paginação. Inicialmente, todos esses dados em conjunto compõem o estado atual do processo no módulo de *threads* e de memória e encapsulam a execução de um processo dentro do *cluster*. Futuramente, introduziremos o módulo de comunicação junto de um processo monitor para manter as conexões consistentes.

Como aplicação direta da virtualização do processo, a migração de processos torna-se mais eficiente. Ao eliminar a necessidade de descobrir quais são e onde estão as dependências que um processo possui, facilitamos a cópia e transferência de suas informações. Isso só é possível porque os *clusters* possuem uma estrutura de *kernel* idêntica, descartando a necessidade do envio de dados relacionados ao SO. Ao evitar o envio de dados redundantes entre *clusters*, atenuamos o impacto da migração sobre a NoC.

O funcionamento da migração é similar ao *Checkpoint/Restore In Userspace (CRIU)*, ferramenta utilizada por *softwares* de gerenciamento de contêineres como o Docker, porém, essa funcionalidade será suportada pelo próprio SO. A migração acontece da seguinte maneira: (i) a execução do processo em um *cluster* é congelada em um estado consistente; (ii) o contexto do processo é enviado via NoC para outro *cluster* (inclui-se nessa etapa o envio dos segmentos de dados e texto do usuário da UArea e das pilhas de execução); e (iii) a execução do processo é restaurada em outro *cluster*. Esse procedimento é implementado por *daemons* do SO.

4. Resultados

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 médias de 100 replicações de cada experimento para garantir 95% de confiança estatística, resultando em um desvio padrão máximo inferior a 1%.

O experimento de manipulação de *threads* mensura os impactos na criação e

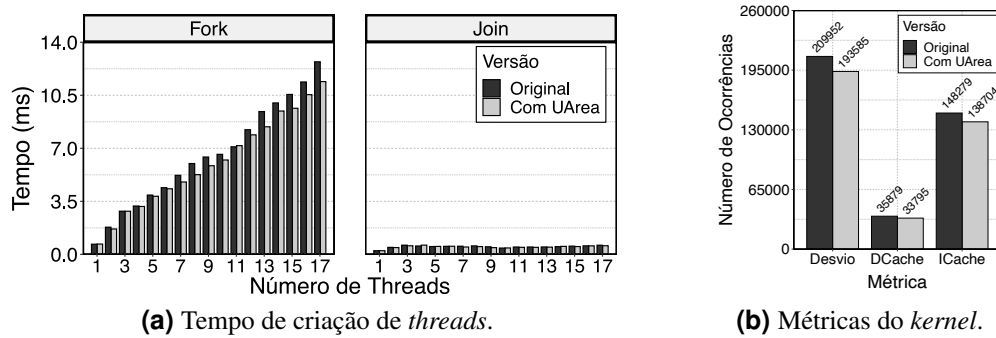


Figura 3. Impactos da virtualização sobre a manipulação de threads.

junção através de diferentes perspectivas. Especificamente, coletamos o tempo de execução, desvios e faltas ocorridas na *cache* de dados e de instrução (Figura 3). Os resultados apresentam um aumento no desempenho das operações de manipulação quando utilizamos a UArea porque exploramos melhor a localidade espacial dos dados, consequentemente, diminuindo 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 tempo médio para migração do usuário foi de 226 ms.

5. Conclusão

Neste trabalho foi explorado um modelo de virtualização leve baseada em contêineres que considera as restrições arquiteturais de *lightweight manycores* para melhorar o suporte de processos em um SO distribuído. Os resultados mostraram que o isolamento das dependências de um processo aumentaram o desempenho de operações do *kernel* e suportaram 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.

Referências

- Castro, M., Franceschini, E., Dupros, F., Aochi, H., Navaux, P. O., and Méhaut, J.-F. (2016). Seismic wave propagation simulations on low-power and performance-centric manycores. *Parallel Computing*, 54:108–120.
- Choudhary, A., Govil, M. C., Singh, G., Awasthi, L. K., Pilli, E. S., and Kapil, D. (2017). A critical survey of live virtual machine migration techniques. *Journal of Cloud Computing*, 6(1):1–41.
- Penna, P. H., Souto, J., Lima, D. F., Castro, M., Broquedis, F., Freitas, H., and Mehaut, J.-F. (2019). On the Performance and Isolation of Asymmetric Microkernel Design for Lightweight Manycores. In *SBESC 2019 - IX Brazilian Symposium on Computing Systems Engineering*, Natal, Brazil.
- Sharma, P., Chaufournier, L., Shenoy, P., and Tay, Y. (2016). Containers and virtual machines at scale: A comparative study. In *Proceedings of the 17th International Middleware Conference*, pages 1–13.
- Thalheim, J., Bhatotia, P., Fonseca, P., and Kasikci, B. (2018). Cntr: Lightweight os containers. In *2018 USENIX Annual Technical Conference*, pages 199–212.