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

Nicolas Vanz[1], **João Vicente Souto**[1], **Márcio Castro**[1]

[1]Laboratório 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 apresentam memória interna limitada e falta de um suporte robusto para virtualização, dificultando o gerenciamento de processos nessas arquiteturas. Neste contexto, este trabalho explora um suporte inicial a uma virtualização leve baseada em contêineres para um sistema operacional distribuído. Os resultados mostram que o método proposto provê uma melhor localidade dos dados e contribui para 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 núcleos de baixa frequência em um único chip; (ii) organizar os núcleos em 20 conjuntos (*clusters*) para compartilhamento de recursos locais; (iii) utilizar 2 *Networks-on-Chip* (NoCs) para transferência de dados entre *clusters*; (iv) possuir um sistema de memória distribuída composto por pequenas memórias locais, e.g., *Static Random Access Memory* (SRAM) de 2 MB; (v) não dispõe 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 redução do consumo energético restringem os tipos de virtualização suportados. Neste contexto, o presente trabalho explora um modelo de virtualização baseado em contêineres adaptado para *lightweight manycores*. 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].

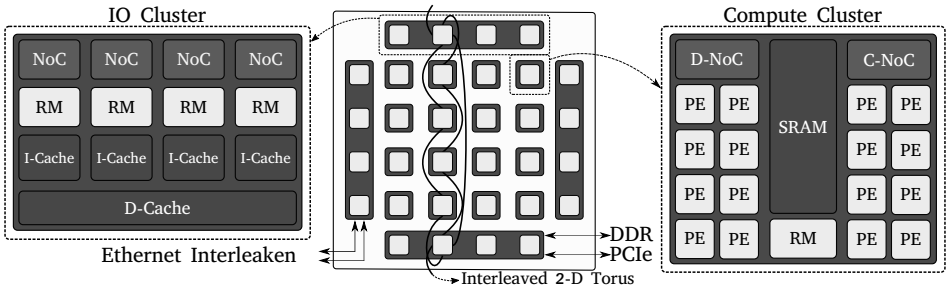


Figura 1. Visão arquitetural do processador Kalray MPPA-256 [Penna et al. 2019].

2. Sistema Operacional Nanvix

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 abstração. Do nível de abstração mais baixo ao mais alto, são elas: (i) *Hardware Abstraction Layer (HAL)* abstrai os recursos de *hardware* sobre uma visão comum. (ii) *Microkernel* gerencia os recursos de um *cluster* e fornece serviços básicos do SO, e.g., *threads*. (iii) *Multikernel* provê serviços complexos do SO de uma forma distribuída através de um modelo cliente-servidor, e.g., memória distribuída.

Em sua abordagem original, os processos no Nanvix são isolados e estaticamente alocados a um único *cluster*. Desta forma, todas as informações do processo são dependentes do *cluster* que o executa. Por exemplo, comunicações são atreladas aos *clusters* envolvidos e não aos processos. A falta de isolamento e virtualização dos recursos dos processos impõe restrições à utilização do processador, por exemplo, afetando o suporte a multi-aplicação, algo necessário a um SO de propósito geral. Melhorar a mobilidade e a disposição dos processos no processador possibilitaria a utilização mais inteligente dos recursos do mesmo. Por exemplo, reduzir o consumo energético aproximando processos que se comunicam intensamente.

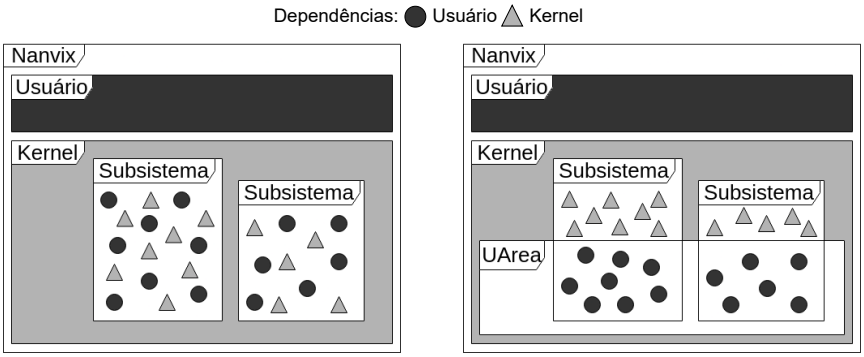
3. Virtualização e Migração de Processos

Visando aumentar a independência dos processos no processador, este trabalho explora um modelo de virtualização leve baseada em contêineres. Nesse modelo, o processo é definido pelos recursos lógicos que compõem o contêiner e não mais pelos recursos físicos e.g., interfaces NoC dos *clusters*. Este trabalho é o resultado de um suporte inicial a esse modelo e engloba apenas o subsistema de *threads*.

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 conteinerização, 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

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



(a) SO sem isolamento. **(b)** SO com isolamento.

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

distinção entre usuário e SO ocorre pela separação das instruções e dados de cada um em segmentos de memória diferentes. Além disso, isolamos as dependências internas do processo em uma área de memória bem definida, separada das demais estruturas 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) *threads* ativas (identificadores e contextos); (ii) ponteiros para suas pilhas de execução; e (iii) variáveis de controle e filas de escalonamento. Inicialmente, a UArea em conjunto com o segmento de dados e instruções do usuário compõem o estado atual do usuário, encapsulando a execução de um processo dentro do *cluster*. Futuramente, introduziremos o módulo de memória e comunicação junto de um processo monitor para manter as conexões consistentes.

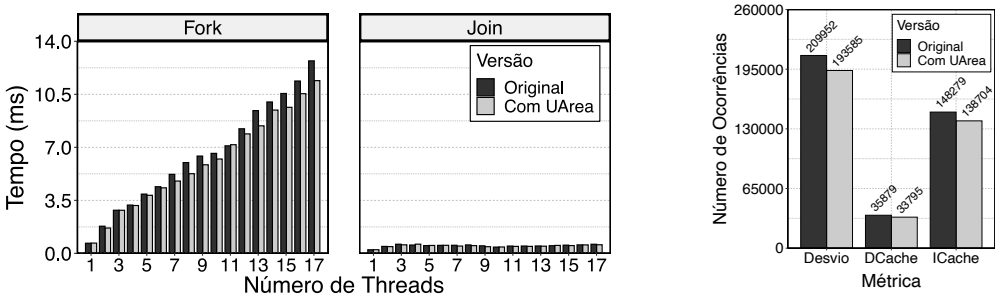
Como aplicação direta do isolamento do processo, a migração de processos torna-se mais eficiente. Com a criação de uma instância isolada do espaço do usuário via conteinerização, eliminamos a necessidade de descobrir quais são e onde estão as dependências do processo, facilitando a transferência de seu contexto. 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.

A funcionalidade inicial de migração 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 *deamons* do 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 código do usuário, da UArea e das pilhas de execução); e (iii) a execução do processo é restaurada em outro *cluster*. Trata-se de uma *hot migration*, em que a aplicação é migrada durante sua execução, com cópia das páginas de memória da aplicação.

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



(a) Tempo de criação de *threads*.

(b) Métricas do *kernel*.

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 da aplicação 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., Francesquini, 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.