Mecanismos de Comunicação entre Clusters para Processadores Lightweight Manycores no Nanvix OS

João Vicente Souto[1], **Pedro H. Penna**[2], **Márcio Castro**[1]

[1] Laboratório de Pesquisa em Sistemas Distribuídos (LaPeSD)
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, Brasil

[2]Université Grenoble Alpes (UGA) – Grenoble, França

joao.vicente.souto@grad.ufsc.br,
pedro.penna@univ-grenoble-alpes.fr, marcio.castro@ufsc.br

Resumo. *Ambientes de desenvolvimento para processadores lightweight manycores pecam em prover uma boa relação entre programabilidade e portabilidade. Neste contexto, este artigo propõe mecanismos de comunicação entre clusters para um sistema operacional distribuído que sejam precisos, fáceis de usar, escalonáveis e facilmente portáveis. Os resultados mostram ser possível suportar algoritmos de comunicação colectiva de forma eficientemente.*

1. Introdução

A próxima grande barreira de desempenho dos sistemas computacionais modernos será proveniente da relação entre poder de processamento e consumo energético. Neste âmbito, processadores *lightweight manycore* surgiram para prover alto nível de paralelismo com baixo consumo energético. Entretanto, o desenvolvimento de aplicações para essa classe de processadores exibe diversos desafios [Castro et al. 2016].

A Figura 1 ilustra as particularidades que diferem os *lightweight manycores* dos *multicores* e *manycores* tradicionais. Especificamente, eles: (i) integram centenas de núcleos de baixo potência agrupados em *clusters*, (ii) são projetados para lidar com cargas de trabalho *Multiple Instruction Multiple Data* (MIMD), (iii) apresentam memória distribuída através de restritivas memórias locais e falta de coerência de cache, (iv) dependem de uma *Network-on-Chip* (NoC) para comunicação entre *clusters*, forçando o uso de uma comunicação híbrida entre memória compartilhada e troca de mensagens, e (v) frequentemente possuem componentes heterogêneos.

Parte dos desafios encontrados ao se trabalhar com *lightweight manycores* deriva diretamente dos *runtimes* e Sistemas Operacionais (SOs) existentes que não lidam corretamente com suas particularidades. Deste modo, eles tornam o ambiente de desenvolvimento mais oneroso e sucetível a erros. Neste contexto, o presente trabalho está incluso na pesquisa e desenvolvimento de um sistema operacional distribuído para *lightweight manycores*. A nossa contribuição é a proposta de mecanismos de comunicação entre clusters que buscam ser mais precisos, fáceis de usar, escalonáveis, e facilmente portáveis.

O restante do trabalho está organizado da seguinte maneira. Na Seção 2, nós apresentamos o projeto Nanvix. A Seção 3 apresenta o *lightweight manycore* utilizado como caso de uso. Em seguida, a Seção 4 comenta os trabalhos relacionados. A Seção 5 e Seção 6 apresentam os mecanismos propostos e a sua avaliação, respectivamente. Por fim, a Seção 7 apresenta as conclusões.

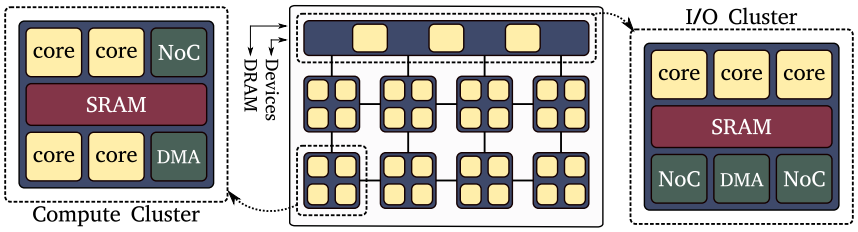


Figura 1. Visão geral de um processador *lightweight manycore*

2. Nanvix Operating System

O Nanvix é um projeto de código aberto e colaborativo que atende a ausência de SOs que lidem com as particularidades dos *lightweight manycore*. Nosso objetivo é desenvolver um SO de propósito geral que melhor relacione programabilidade e portabilidade e seja compatível com o padrão *Portable Operating System Interface* (POSIX), chamado *Nanvix OS*. Ele adota uma estrutura *multikernel*, onde os serviços do SO são processos que rodam isoladamente atendendo requisições de processos de usuário através da troca de mensagens. Dentro de um *cluster* é utilizado um *microkernel* assíncrono para amenizar a interferência entre os núcleos [Penna et al. 2019b]. Na camada mais baixa, o Nanvix exporta uma visão padronizada e concisa desses processadores através de uma Camada de Abstração de Hardware (HAL) [Penna et al. 2019a]. O presente trabalho está incluso nas camadas do *microkernel* e HAL.

3. Processador Lightweight Manycore MPPA-256

O Kalray MPPA-256 [de Dinechin et al. 2013] é uma das arquiteturas suportadas pelo Nanvix OS e foi utilizada como caso de uso. Especificamente, ele integra 288 núcleos de propósito geral, agrupados em 16 Clusters de Computação (CCs), destinados a computação útil, e 4 Clusters de I/O (IOs) responsáveis pela comunicação com periféricos. Para comunicação entre *clusters*, o Kalray MPPA-256 apresenta uma NoC para dados e outra para comandos, denominadas *Data Network-on-Chip* (D-NoC) e *Control Network-on-Chip* (C-NoC) respectivamente. Cada uma das NoCs apresenta características distintas, destacando-se as diferentes quantidades de recursos de comunicação (RX e TX) e a quantidade de dados transmitidos (apenas 64 bits de cada vez pela C-NoC).

4. Trabalhos Relacionados

Kluge et al. [Kluge et al. 2014] projetaram um SO que provê canais de comunicação unidirecionais com suporte a configuração de políticas de Qualidade de Serviço (QoS), argumentando que é possível suportar abstrações mais complexas sobre esses canais. *Wentzlaff et al.* [Wentzlaff et al. 2011], por sua vez, fornece a comunicação entre processos através da troca de mensagens baseado na abstração de *mailbox*, onde justificam que esse tipo de comunicação é facilmente portável e escalonável.

5. Mecanismos de Comunicação entre Clusters

Os mecanismos de comunicação entre *clusters* são constituídos de três abstrações, *Sync*, *Mailbox* e *Portal*. A definição dessas abstrações generalizam três comportamentos que frequentemente aparecem em sistemas distribuídos, i.e., sincronizações, trocas de mensagens de controle e grandes trocas de dados. Desta forma, exportamos uma visão abstrata e padronizada dos recursos de comunicação existentes nas mais diversas arquiteturas.

Primeiramente, a abstração *Sync* provê a criação de barreiras distribuídas. Seu comportamento se assemelha ao POSIX *Signals*. Existem dois modos de sincronização. Primeiro, o modo `ALL_TO_ONE` define que um *cluster* mestre deve aguardar N notificações de N escravos. Segundo, no modo `ONE_TO_ALL` o mestre notifica N escravos, liberando-os do bloqueio. A implementação no Kalray MPPA-256 utilizou apenas recursos da C-NoC, onde o principal argumento é a lista dos *clusters* envolvidos. A primeira posição nesta lista deve ser ocupada obrigatóriamente pelo mestre, a partir do qual será inferido quais recursos de hardware deverão ser utilizados. Deste modo, é possível abstrair o conhecimento e manipulação do hardware pelo usuário.

Segundo, a abstração *Mailbox* provê a troca de mensagens de tamanho fixo. Ela é similar ao POSIX Message Queue, onde o receptor alocá espaço suficiente para receber N mensagens. O emissor, por sua vez, envia uma mensagem para um local pré-determinado. O comportamento da *Mailbox* define um controle de fluxo permitindo a emissão de apenas uma mensagem de cada vez. Quando o receptor consumir uma mensagem, ele notificará o emissor que a enviou. A implementação utilizou recursos da C-NoC e da D-NoC. A fila de mensagens foi alocada no espaço de memória do kernel para abstrair o controle e manipulação das mensagens do usuário.

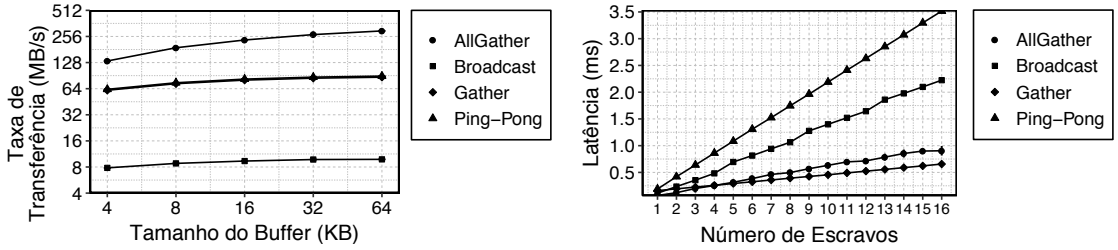
Por fim, a abstração *Portal* permite a troca de quantidades arbitrárias de dados entre dois *clusters*. Similar ao POSIX Pipe, a comunicação é unidirecional. O *Portal*, assim como a *Mailbox*, implementa um controle de fluxo para garantir a QoS. Neste controle, o emissor só enviará os dados quando o receptor estiver apto a receber. Deste modo, a implementação do *Portal*, utilizando recursos da C-NoC e D-NoC, eliminou a necessidade de cópias intermediárias, onde a comunicação é configurada com endereços de memória do próprio espaço de usuário.

6. Resultados Experimentais

Para avaliar os serviços de transferência de dados, foram elaboradas quatro rotinas de comunicação coletiva que reproduzem comportamentos existentes no Nanvix OS, i.e., *Broadcast*, *Gather*, *AllGather* e *Ping-Pong* [Wickramasinghe and Lumsdaine 2016]. Para garantir 95% de confiança, foram realizadas 50 replicações, descartando-se as primeiras 10 para ignorar o período de aquecimento e conduzindo a um erro padrão inferior a 1%.

Os experimentos do Portal avaliaram a taxa de transferência da comunicação. Os parâmetros mantiveram constante o número de *clusters* envolvidos (1 IOs e 16 CCs), variando a quantidade de dados transmitidos (4 KB até 64 KB). Os resultados na Figura 2(a) exibem três comportamentos. Primeiro, devido ao gargalo do único emissor, o *Broadcast* apresentou os piores resultados. Segundo, o *Gather* e *Ping-Pong* mostram resultados similares porque o mestre da operação dita o fluxo das comunicações devido ao QoS. Por último, o *AllGather* obteve os melhores resultados porque se benificia do paralelismo das comunicações. No contexto do Nanvix OS, é possível inferir que os tamanhos entre 8 KB e 16 KB favorecem a taxa de transferência do *Portal* no uso pelos subsistemas do SO.

Os experimentos da Mailbox avaliaram a latência da comunicação. Foi mantido constante o tamanho da mensagem (120 B) e variou-se o número de *clusters* envolvidos (1 IOs e 1 à 16 CCs). A Figura 2(b) também apresentaram três comportamentos. Primeiro, o *Gather* e *AllGather* se benificiaram do recebimento paralelo das mensagens e obtiveram as menores latências. Segundo, o comportamento do *Broadcast* sofre do mesmo problema do *Portal*. Por último, apesar do *Ping-Pong* também se beneficiar do recebimento paralelo, o mestre deve atender sequencialmente cada uma das mensagens,



(a) *Portal* (b) *Mailbox*

Figura 2. Resultados Experimentais.

por isso apresentou os piores resultados. De forma geral, os resultados mostraram como algoritmos distribuídos podem ser suportados eficientemente no Nanvix OS e destacam possíveis melhorias.

7. Conclusão

Neste trabalho, foi proposto mecanismos de comunicação entre *cluster* para processadores *lightweight manycores* no Nanvix OS. Os resultados mostraram como algoritmos distribuídos bem conhecidos podem ser eficientemente suportados pelo Nanvix OS. Como trabalhos futuros no contexto do Nanvix OS, pretende-se otimizar a implementação do Kalray MPPA-256, realizar o porte do *Message Passing Interface* (MPI), estudar sistemas de paginação e escalonamento de processos distribuídos.

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.

de Dinechin, B. D., de Massas, P. G., Lager, G., Léger, C., Orgogozo, B., Reybert, J., and Strudel, T. (2013). A Distributed Run-Time Environment for the Kalray MPPA-256 Integrated Manycore Processor. In *Procedia Computer Science*, volume 18 of *ICCS '13*, pages 1654–1663, Barcelona, Spain. Elsevier.

Kluge, F., Gerdes, M., and Ungerer, T. (2014). 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*, ISORC '14, pages 238–245, Reno, Nevada, USA. IEEE.

Penna, P. H., Francis, D., and Souto, J. (2019a). 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.

Penna, P. H., Souto, J., Lima, D. F., Castro, M., Broquedis, F., Freitas, H., and Mehaut, J.-F. (2019b). On the Performance and Isolation of Asymmetric Microkernel Design for Lightweight Manycores. In *SBESC 2019 - IX Brazilian Symposium on Computing Systems Engineering*, Natal, Brazil.

Wentzlaff, D., Gruenwald, C., Beckmann, N., Belay, A., Kasture, H., Modzelewski, K., Youseff, L., Miller, J., and Agarwal, A. (2011). Fleets: Scalable services in a factored operating system.

Wickramasinghe, U. and Lumsdaine, A. (2016). A survey of methods for collective communication optimization and tuning. *CoRR*, abs/1611.06334.