An OS Service for Transparent Remote Memory Accesses in NoC-Based Lightweight Manycores

Pedro Henrique Penna, Matheus Souza, Emmanuel Podestá Júnior, Bruno Nascimento, Márcio Castro, François Broquedis, Henrique Freitas, Jean-François Méhaut

An OS Service for Transparent Remote Memory Accesses in NoC-Based Lightweight Manycores

Pedro Henrique Penna [1 2], Matheus Souza [2], Emmanuel P. Júnior [3], Bruno Nascimento [3], Márcio Castro [3], François Broquedis [4], Henrique Freitas [2] and Jean-François Méhaut [1]

Introduction

- **Lightweight Manycores Are Substantially Different**
  - Integrate up to thousands of simple and low-power cores
  - Feature rich, fast and reliable interconnects
  - Present a constrained distributed memory configuration
- **Current Runtime Systems Miss Rich Abstractions**
  - The engineer should implement all by himself
  - A fully-featured OS would make software design easier

Goals and Contributions

- **Target Challenges That Arise from the Distributed Memory**
  - Data accessing, tiling and migration
  - Address space expansion
  - Secure data sharing
- **Propose the Remote Memory (RMem) Service**
  - New OS facility that provides a shared memory abstraction
- **Introduce Communication Primitives on Top of RMem**
  - Rely on a one-sided programming paradigm
  - Enable applications to share data in a secure fashion
- **Present a Prototype of RMem for the MPPA-256 Processor**
  - Integration with Nanvix (`https://github.com/nanvix`)

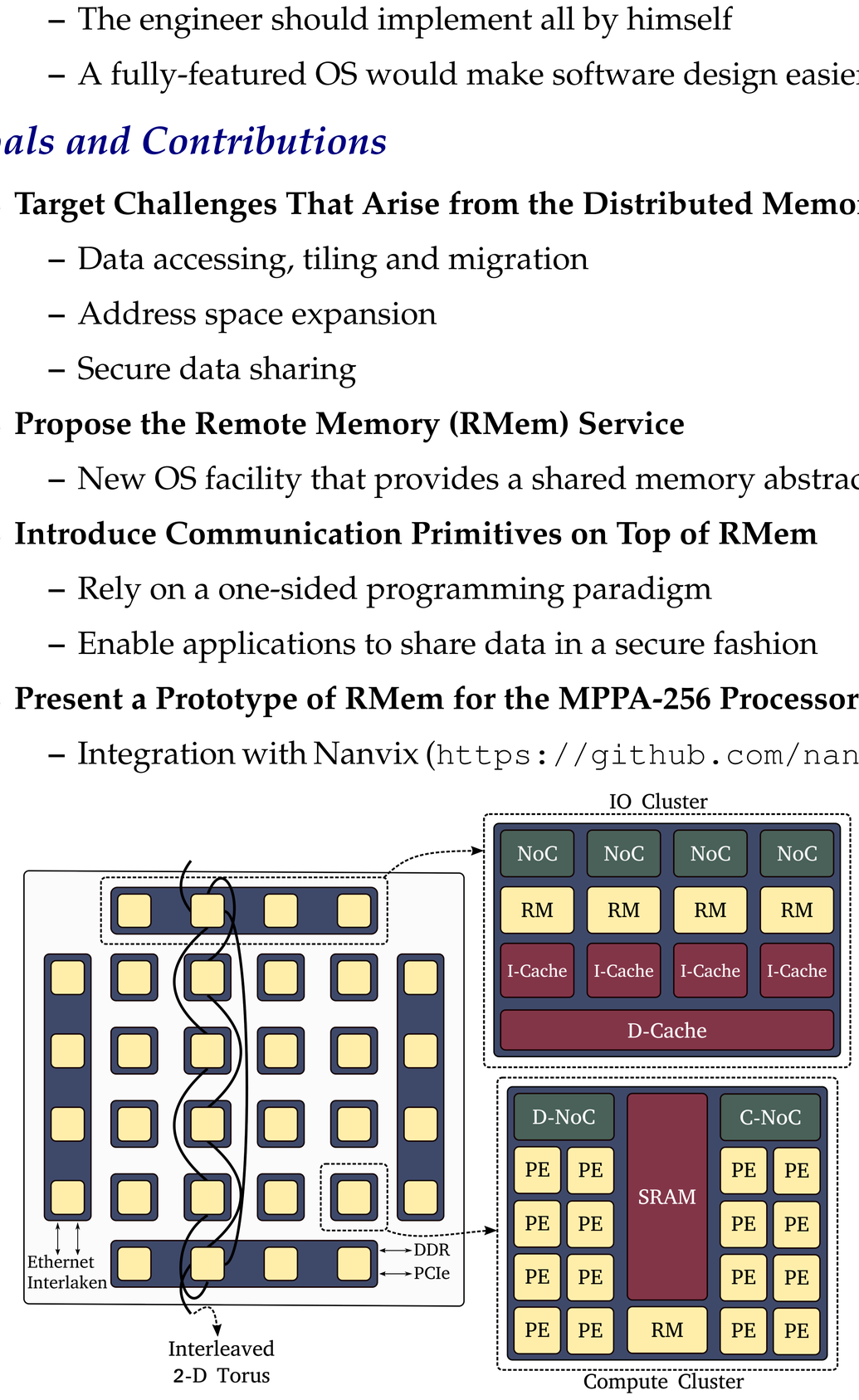

Figure 1: Architectural overview of MPPA-256.

The RMem Service

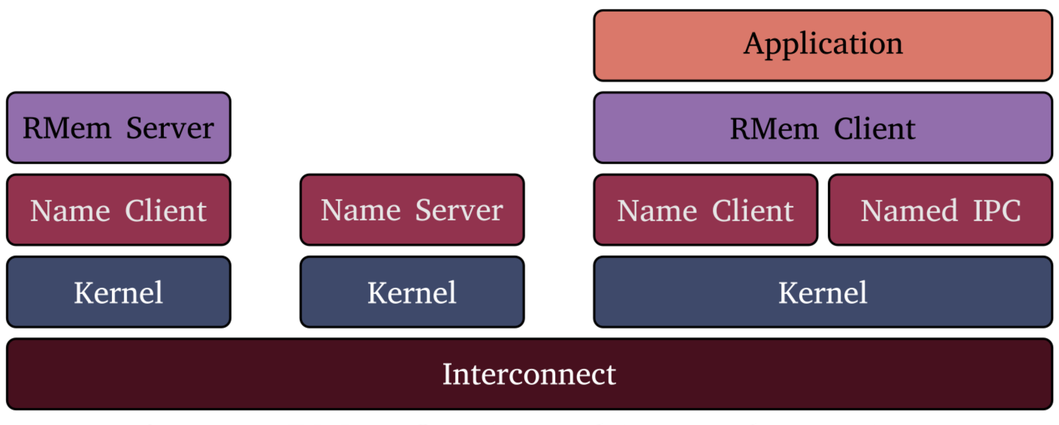

Figure 2: RMem Service architectural overview.

- **Name Service**: provides name resolution protocol
- **Named IPC**: mailbox (`1:N`) and portal (`M:N`)

`memread(void *local, off_t remote, size_t size)`

1. Parse the remote address
2. Resolve location of target RMem server
3. Send read request to the server through a mailbox
4. Enable remote portal reads from the RMem server
5. Receive data from the RMem server via a portal

`memwrite(void *local, off_t remote, size_t size)`

1. Parse the remote address
2. Resolve location of the target RMem server
3. Send write request to the server through a mailbox
4. Send data to the RMem server via a portal

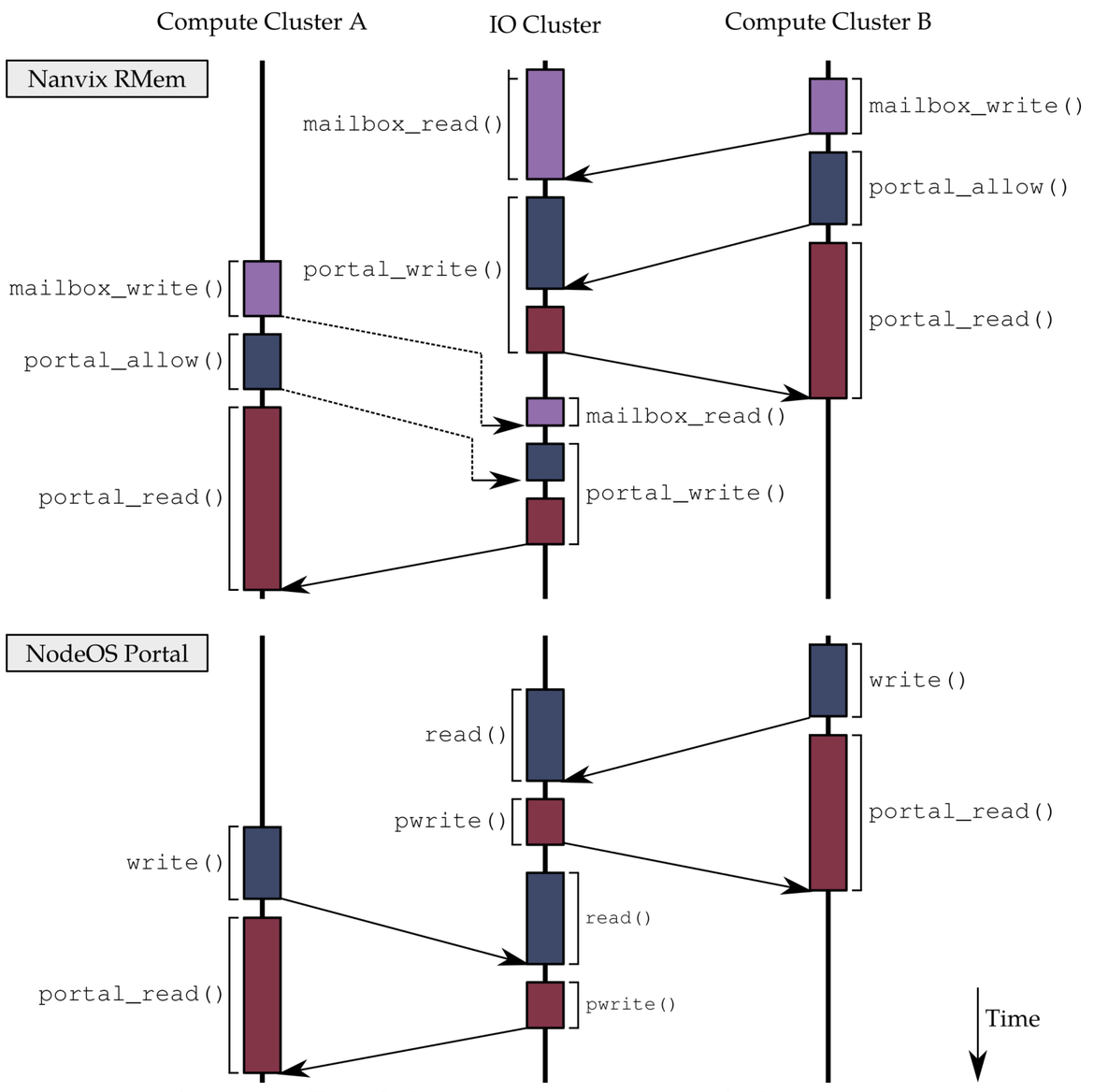

Figure 3: Breakdown of remote reads by 2 peers.

Experimental Results

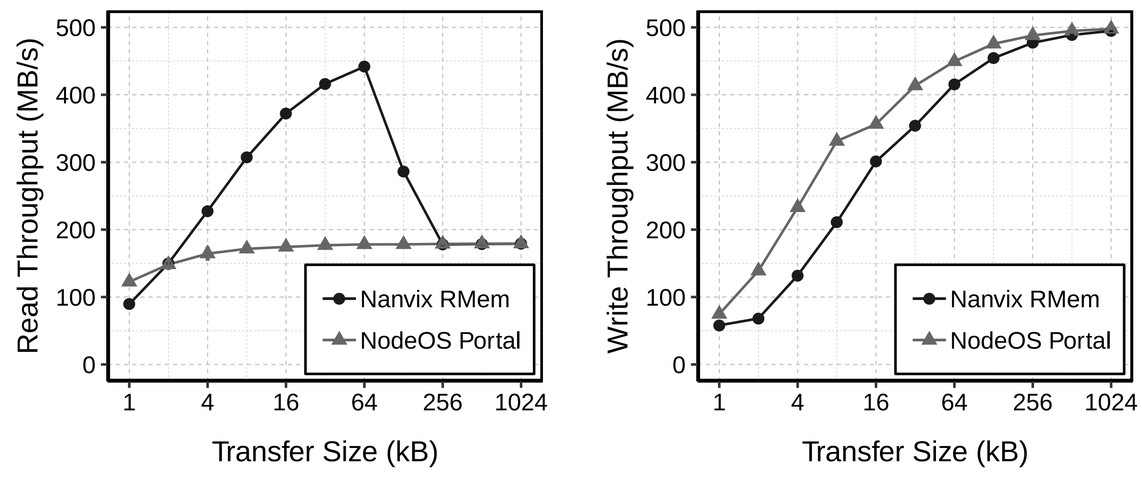

Figure 4: Experimental results for synthetic kernel.

Conclusions

- RMem Service and NodeOS have similar write performance
- Read protocol maximizes concurrency
- Results encourage a native implementation of our service

[1]Université Grenoble Alpes (UGA), Grenoble, France
[2]Pontifícia Universidade Católica de Minas Gerais (PUC Minas), Brazil
[3]Universidade Federal de Santa Catarina (UFSC), Brazil
[4]Institut National Polytechnique de Grenoble (Grenoble INP), France