



HAL
open science

On the Performance and Isolation of Asymmetric Microkernel Design for Lightweight Manycores

Pedro Henrique Penna, João Souto, Davidson Francis Lima, Márcio Castro,
François Broquedis, Henrique H Freitas, Jean-François Mehaut

► **To cite this version:**

Pedro Henrique Penna, João Souto, Davidson Francis Lima, Márcio Castro, François Broquedis, et al.. On the Performance and Isolation of Asymmetric Microkernel Design for Lightweight Manycores. SBESC 2019 - IX Brazilian Symposium on Computing Systems Engineering, Nov 2019, Natal, Brazil. pp.1-31. hal-02297637v2

HAL Id: hal-02297637

<https://hal.science/hal-02297637v2>

Submitted on 22 Nov 2019 (v2), last revised 30 Mar 2020 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

On the Performance and Isolation of Asymmetric Microkernel Design for Lightweight Manycores

Pedro Henrique Penna^{1,2}, João Vicente Souto³, Davidson Francis Lima²,
Márcio Castro³, François Broquedis⁴,
Henrique Freitas² and Jean-François Méhaut¹

¹Université Grenoble Alpes (UGA)

²Pontifícia Universidade Católica de Minas Gerais (PUC Minas)

³Universidade Federal de Santa Catarina (UFSC)

⁴Institut National Polytechnique de Grenoble (Grenoble INP)

SBESC '19



Introduction

Lightweight (LW) Manycores – Overview

- Hundreds of Lightweight Cores
 - Target MMID computing workloads
 - Expose massive thread-level parallelism
 - Feature low-power consumption
- Distributed Memory Architecture
 - Grants scalability
 - Delivers predictability
- On-Chip Heterogeneity
 - Enables adaptability to diverse workloads
 - Uncovers high-energy efficiency
- Rich On-Chip Interconnects
 - Offer quality of service
 - Allow asynchronous communications

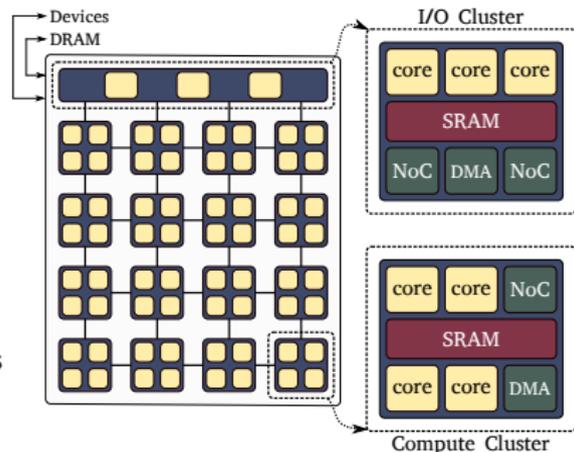


Figure: Overview of a manycore.

Introduction

Lightweight (LW) Manycores – Overview

- Hundreds of Lightweight Cores
 - Target MMIO computing workloads
 - Expose massive thread-level parallelism
 - Feature low-power consumption
- Distributed Memory Architecture
 - Grants scalability
 - Delivers predictability
- On-Chip Heterogeneity
 - Enables adaptability to diverse workloads
 - Uncovers high-energy efficiency
- Rich On-Chip Interconnects
 - Offer quality of service
 - Allow asynchronous communications

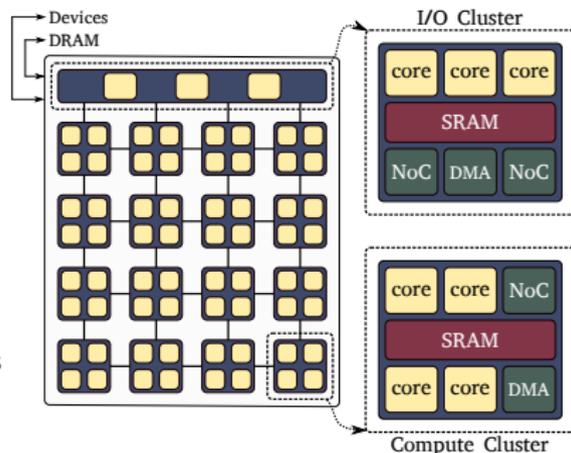


Figure: Overview of a manycore.

It is a distributed architecture in a chip!

Introduction

Lightweight (LW) Manycores – Challenges

- Used in embedded computing and HPC
 - What about multi-application support?
- High Density Circuit Integration
 - Heat dissipation
 - Dark silicon
- Distributed Memory Architecture
 - Data tiling (small local memories)
 - Message passing
- On-Chip Heterogeneity
 - Thread scheduling and placement
- Rich On-Chip Interconnects
 - Network congestion
 - Security checking

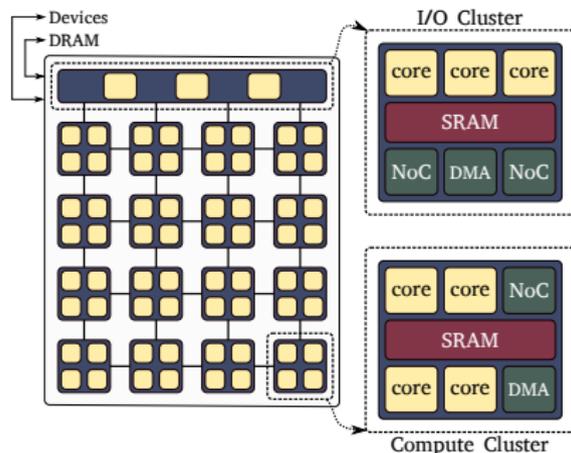


Figure: Overview of a manycore.

Introduction

Lightweight (LW) Manycores – Challenges

- Used in embedded computing and HPC
 - What about multi-application support?
- High Density Circuit Integration
 - Heat dissipation
 - Dark silicon
- Distributed Memory Architecture
 - Data tiling (small local memories)
 - Message passing
- On-Chip Heterogeneity
 - Thread scheduling and placement
- Rich On-Chip Interconnects
 - Network congestion
 - Security checking

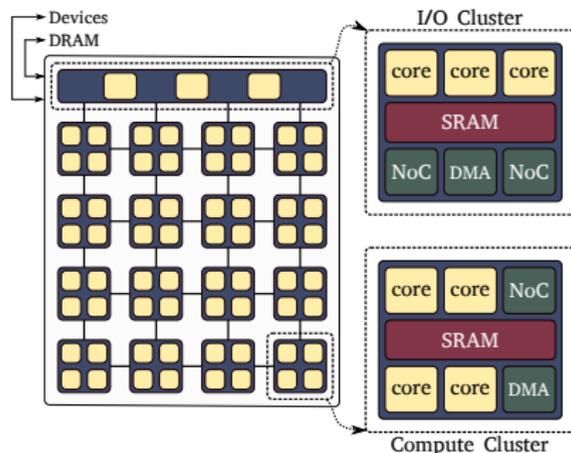


Figure: Overview of a manycore.

Performance vs Programmability vs Portability

Introduction

Lightweight (LW) Manycores – Operating System Support

- OSes enhance programmability and portability
 - Expose rich abstractions and APIs
 - Multiplex resources and ensure policies

Introduction

Lightweight (LW) Manycores – Operating System Support

- OSes enhance programmability and portability
 - Expose rich abstractions and APIs
 - Multiplex resources and ensure policies
- How about commodity operating systems?
 - Ex: Linux, FreeBSD, Windows...
 - Pros: instantaneous support for tons of software
 - Symmetric design leads to **cache interference** (Wentzlauff and Agarwal 2009)
 - Poor fine-grain **lock scalability** (Amdahl's Law)
 - Increasingly **diverse hardware** (Barbalace et al. 2015)

Introduction

Lightweight (LW) Manycores – Operating System Support

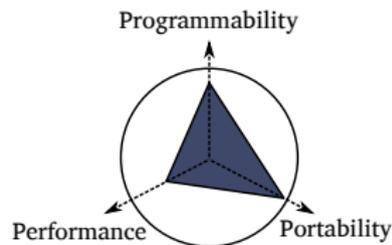
- OSES enhance programmability and portability
 - Expose rich abstractions and APIs
 - Multiplex resources and ensure policies
- How about commodity operating systems?
 - Ex: Linux, FreeBSD, Windows...
 - Pros: instantaneous support for tons of software
 - Symmetric design leads to **cache interference** (Wentzlaff and Agarwal 2009)
 - Poor fine-grain **lock scalability** (Amdahl's Law)
 - Increasingly **diverse hardware** (Barbalace et al. 2015)
- How about distributed operating systems?
 - Ex: microkernels and multikernels
 - Pros: modularity and scalability
 - Miss **support for rich on-chip interconnects** (Dinechin et al. 2013)
 - Do not cope with **small local memories** (Olofsson, Nordstrom, and Ul-Abdin 2014)

- OSeS enhance programmability and portability
 - Expose rich abstractions and APIs
 - Multiplex resources and ensure policies
- How about commodity operating systems?
 - Ex: Linux, FreeBSD, Windows...
 - Pros: instantaneous support for tons of software
 - Symmetric design leads to **cache interference** (Wentzlauff and Agarwal 2009)
 - Poor fine-grain **lock scalability** (Amdahl's Law)
 - Increasingly **diverse hardware** (Barbalace et al. 2015)
- How about distributed operating systems?
 - Ex: microkernels and multikernels
 - Pros: modularity and scalability
 - Miss **support for rich on-chip interconnects** (Dinechin et al. 2013)
 - Do not cope with **small local memories** (Olofsson, Nordstrom, and Ul-Abdin 2014)

Existing OSeS do not address lightweight manycores!

■ **Long-Term Goal:** Propose an OS for LW Manycores

- Deliver portability across multiple platforms
- Expose a POSIX-compliant interface
- Provide flexible view of the platform
- Embrace a multikernel OS structure
- Rely on asymmetric microkernels as building blocks

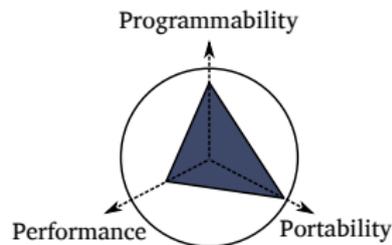


■ **Goal of This Work:** Assess an Asymmetric Microkernel Design for LW Manycores

- Microkernel Structure: improves flexibility and portability
- Asymmetric Design: delivers scalability

■ **Long-Term Goal:** Propose an OS for LW Manycores

- Deliver portability across multiple platforms
- Expose a POSIX-compliant interface
- Provide flexible view of the platform
- Embrace a multikernel OS structure
- Rely on asymmetric microkernels as building blocks



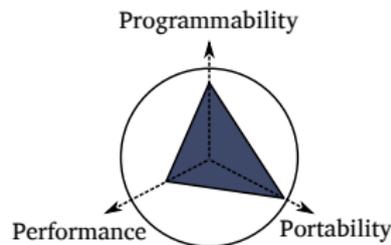
■ **Goal of This Work:** Assess an Asymmetric Microkernel Design for LW Manycores

- Microkernel Structure: improves flexibility and portability
- Asymmetric Design: delivers scalability

■ **Scientific Contribution:** Insights on Kernel Construction for LW Manycores

- Quantitative results on performance and isolation of the assessed design
- Discussion on co-design aspects between the OS kernel and the hardware

- **Long-Term Goal:** Propose an OS for LW Manycores
 - Deliver portability across multiple platforms
 - Expose a POSIX-compliant interface
 - Provide flexible view of the platform
 - Embrace a multikernel OS structure
 - Rely on asymmetric microkernels as building blocks
- **Goal of This Work:** Assess an Asymmetric Microkernel Design for LW Manycores
 - Microkernel Structure: improves flexibility and portability
 - Asymmetric Design: delivers scalability
- **Scientific Contribution:** Insights on Kernel Construction for LW Manycores
 - Quantitative results on performance and isolation of the assessed design
 - Discussion on co-design aspects between the OS kernel and the hardware
- **Technical Contribution:** Nanvix Microkernel
 - Open source asymmetric microkernel for LW manycores
 - Supports multiple baremetal platforms (MPPA-256, RISC-V, OpenRISC)



1 Introduction

- Context and Motivation
- Target Problem
- Goal and Contributions

2 The Nanvix OS

- Overview
- System Structure
- Microkernel Overview

3 Experimental Results

- Evaluation Methodology
- Microbenchmarks
- Synthetic Benchmarks

4 Conclusions

1 Introduction

- Context and Motivation
- Target Problem
- Goal and Contributions

2 The Nanvix OS

- Overview
- System Structure
- Microkernel Overview

3 Experimental Results

- Evaluation Methodology
- Microbenchmarks
- Synthetic Benchmarks

4 Conclusions

The Nanvix Operating System

Overview – The Nanvix Project

- Re-Engineered Version of Nanvix to LW Manycores
 - **Home-grown** OS
 - 4 Professors (Brazil and France)
 - 1 PhD, 2 MSc and 2 BSc Students
 - 9 past contributors
 - UGA, PUC Minas, UFSC and Grenoble INP
- Project Guidelines
 - Be Open: invite others to collaborate
 - Be Permissive: enable free adaptability
- Design Principles
 - Be Portable: run on multiple architectures
 - Be Scalable: embrace distributed configuration
 - Be Flexible: expose multiple APIs



Figure: Bingo, our mascot.

<https://github.com/nanvix>

■ Architectural Model

- Cores are grouped into clusters
- Each cluster has its own physical address space
- Intra-Cluster communication: shared memory
- Inter-Cluster communication: NoC

■ Multikernel with Three-Layers

■ Kernels

- One instance on each cluster
- Provide minimum abstractions
- Ensure policies and security

■ System Servers

- Run on top of kernels at user-level
- Provide traditional abstractions
- Collaboratively implement subsystems

■ Runtime Libraries

- Run alongside with user-applications
- Interface with system servers
- Expose standard APIs (i.e. POSIX)

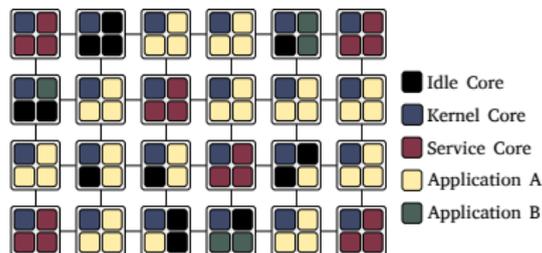


Figure: The multikernel OS structure.

- **Microkernel Design**
 - Asymmetric: runs on a dedicated core of a cluster
 - Small: provides only essential abstractions (about 5k LoC)
 - Portable: supports MPPA-256, RISC-V and OpenRISC based manycores
- **Thread Management System**
 - Non-interruptible kernel threads
 - Sleep/wakeup primitives
 - Exception handling forwarding
 - Thread checkpointing
- **Memory Management System**
 - Single address space
 - Two-level paging scheme
- **IPC Facility**
 - Inter-cluster synchronization
 - Inter-cluster communication

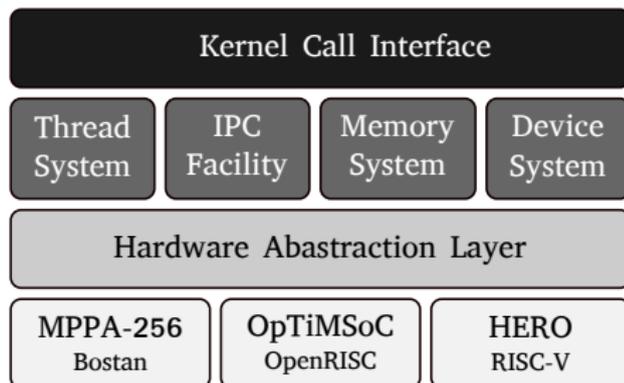


Figure: An overview of the Nanvix kernel.

1 Introduction

- Context and Motivation
- Target Problem
- Goal and Contributions

2 The Nanvix OS

- Overview
- System Structure
- Microkernel Overview

3 Experimental Results

- Evaluation Methodology
- Microbenchmarks
- Synthetic Benchmarks

4 Conclusions

■ Microbenchmark Experiments

- Assess asymmetric design
- L-Kcall: performance of local kernel calls
- R-Kcall: performance of remote kernel calls

■ Synthetic Benchmark Experiments

- Evaluate performance on representative use cases
- Fork-Join: scalability for fork-join programming model
- KNoise: kernel interference on application execution

■ Microbenchmark Experiments

- Assess asymmetric design
- L-Call: performance of local kernel calls
- R-Call: performance of remote kernel calls

■ Synthetic Benchmark Experiments

- Evaluate performance on representative use cases
- Fork-Join: scalability for fork-join programming model
- KNoise: kernel interference on application execution

■ Experimental Platform

- Compute Cluster of MPPA-256 Boston (16 cores, nooc 2 MB memory)
- Kalray Accesscore 2.8.1 (Hypervisor 1.0, GCC 4.9.4 & Binutils 2.11.0)

■ Evaluation Methodology

- Full factorial design for each experiment (70 configurations in total)
- 30 replicas for each experimental configuration ($< 1\%$ of c.o.v)
- Nanvix Microkernel 0a0088b build with -O3 flags (128 kb memory footprint)

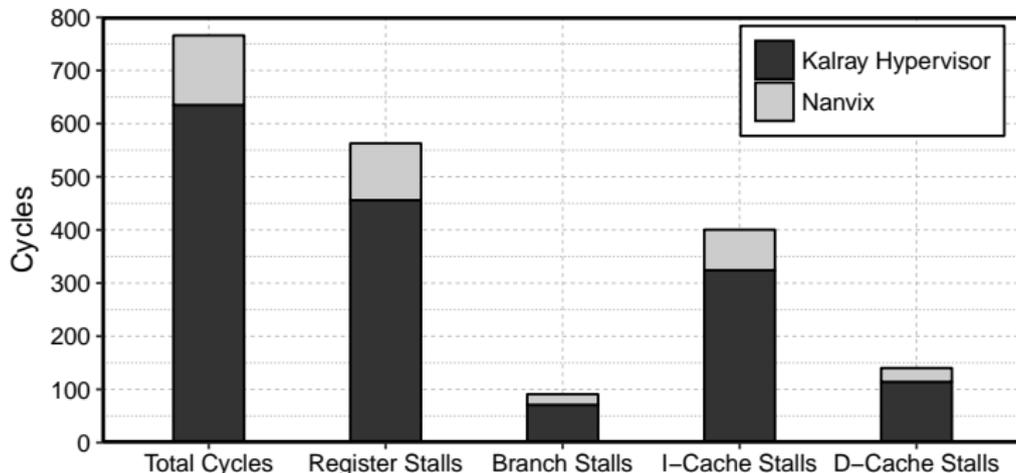


Figure: Breakthrough of local kernel calls in Nanvix.

- About 164 cycles are required for mode switch (i.e.: user to kernel)

The Nanvix Operating System

Microkernel – L-KCall Benchmark Results

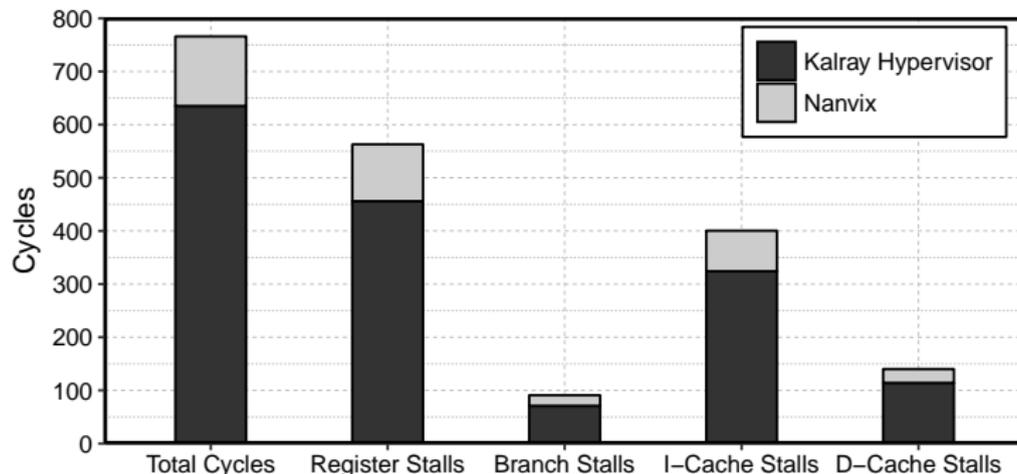


Figure: Breakthrough of local kernel calls in Nanvix.

- Low kernel interference
 - Complex execution flow does not mess up branch unit
 - D-Cache is not badly impacted (low capacity misses)

The Nanvix Operating System

Microkernel – L-KCall Benchmark Results

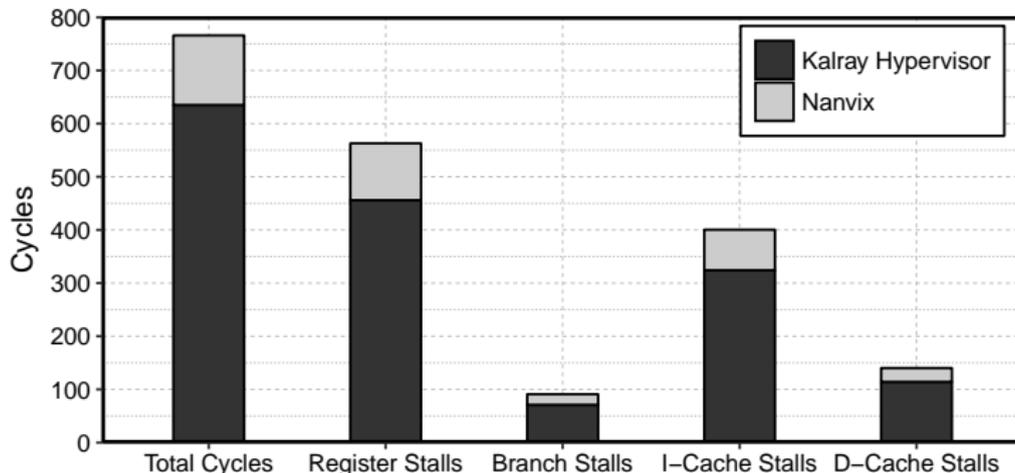


Figure: Breakthrough of local kernel calls in Nanvix.

- I-Cache and I-Fetch units are badly performing
 - Working size set is small enough (less than 8 kB)
 - I-Cache stalls account for 52% of time
- Register file stalls account for 73% of time (bad code generation)

The Nanvix Operating System

Microkernel – R-KCall Benchmark Results

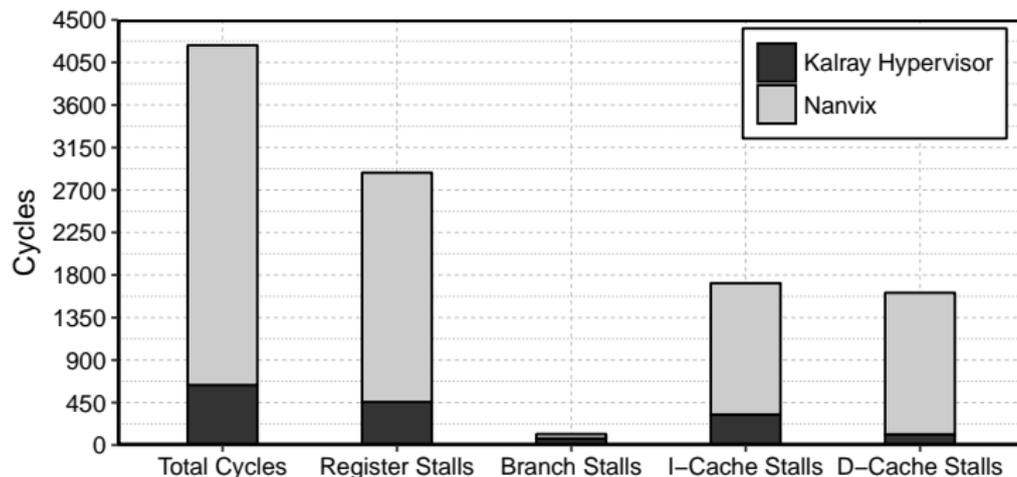


Figure: Breakthrough of remote kernel calls in Nanvix.

- Overheads in inter-core synchronization do matter
 - D-Cache stalls account for 38% of time
 - Hardware cache coherency is not supported
 - Hardware misses elective cache line invalidation

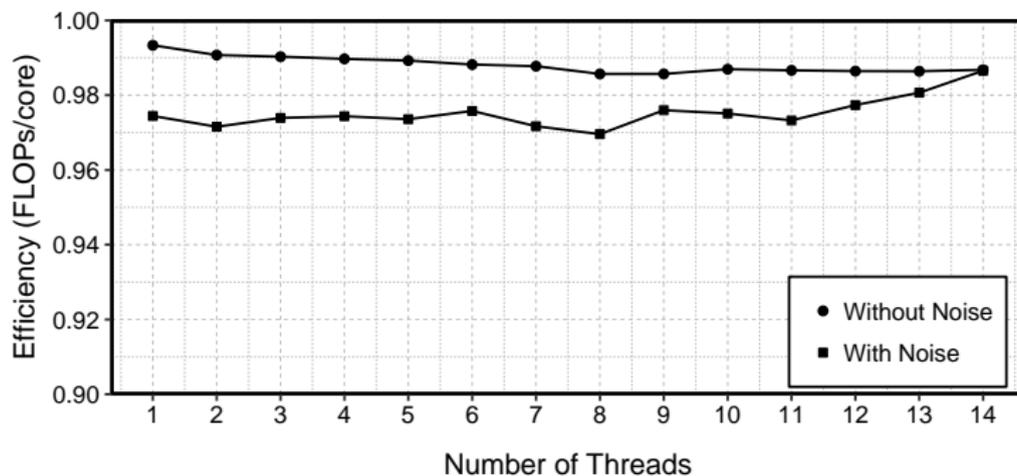


Figure: Kernel noise scalability in Nanvix.

- Linear scalability for user applications
 - Constant overhead of 0.5% per thread (bad-case scenario)

1 Introduction

- Context and Motivation
- Target Problem
- Goal and Contributions

2 The Nanvix OS

- Overview
- System Structure
- Microkernel Overview

3 Experimental Results

- Evaluation Methodology
- Microbenchmarks
- Synthetic Benchmarks

4 Conclusions

Conclusions on Our Research

Motivation, Goal and Contribution

- **Motivation:** Existing OSES Do Not Address LW manycores
 - Distributed architecture with small local memories
 - On-chip heterogeneity
 - Rich on-chip interconnect
- **Log-Term Goal:** Propose an OS for LW Manycores
 - Deliver portability across multiple platforms
 - Provide a POSIX interface and a flexible view of the platform
 - Embrace a multikernel OS structure
 - Rely on asymmetric microkernels as building blocks
- **Goal of This Work:** Assess an Asymmetric Microkernel Design for LW Manycores
- **Scientific Contribution:** Insights on Kernel Construction for LW Manycores
 - Asymmetric microkernel design delivers performance isolation and scalability
 - I-Cache and I-Fetch units are a hotspot for improvement
 - D-Cache coherence or selective cache invalidation may push performance further
- **Technical Contribution:** Nanvix Microkernel
 - Asymmetric microkernel for LW manycores (MPPA-256, RISC-V, OpenRISC)

Thank You!

On the Performance and Isolation of Asymmetric Microkernel Design for Lightweight Manycores

<https://github.com/nanvix>

Pedro Henrique Penna^{1,2}, João Vicente Souto³, Davidson Francis Lima²,
Márcio Castro³, François Broquedis⁴,
Henrique Freitas² and Jean-François Méhaut¹

¹Université Grenoble Alpes (UGA)

²Pontificia Universidade Católica de Minas Gerais (PUC Minas)

³Universidade Federal de Santa Catarina (UFSC)

⁴Institut National Polytechnique de Grenoble (Grenoble INP)





Antonio Barbalace et al. “Popcorn: Bridging the Programmability Gap in Heterogeneous-ISA Platforms”. In: [European Conf. on Computer Systems](#). Bordeaux, France, Apr. 2015, pp. 1–16. ISBN: 978-1-4503-3238-5. DOI: 10.1145/2741948.2741962.



Benoit de Dinechin et al. “A Clustered Manycore Processor Architecture for Embedded and Accelerated Applications”. In: [Int. Conf. on High Performance Extreme Computing](#). Waltham, USA, 2013, pp. 1–6. ISBN: 978-1-4799-1365-7. DOI: 10.1109/HPEC.2013.6670342.



Andreas Olofsson, Tomas Nordstrom, and Zain Ul-Abdin. “Kickstarting High-Performance Energy-Efficient Manycore Architectures with Epiphany”. In: [Asilomar Conf. on Signals, Systems and Computers](#). 2014, pp. 1719–1726. ISBN: 978-1-4799-8297-4. DOI: 10.1109/ACSSC.2014.7094761.



David Wentzlaff and Anant Agarwal. “Factored Operating Systems (FOS): The Case for a Scalable Operating System for Multicores”. In: [ACM SIGOPS Operating Systems Review](#) 43.2 (Apr. 2009), pp. 76–85. ISSN: 0163-5980. DOI: 10.1145/1531793.1531805.

The Nanvix Operating System

Microkernel – Overview

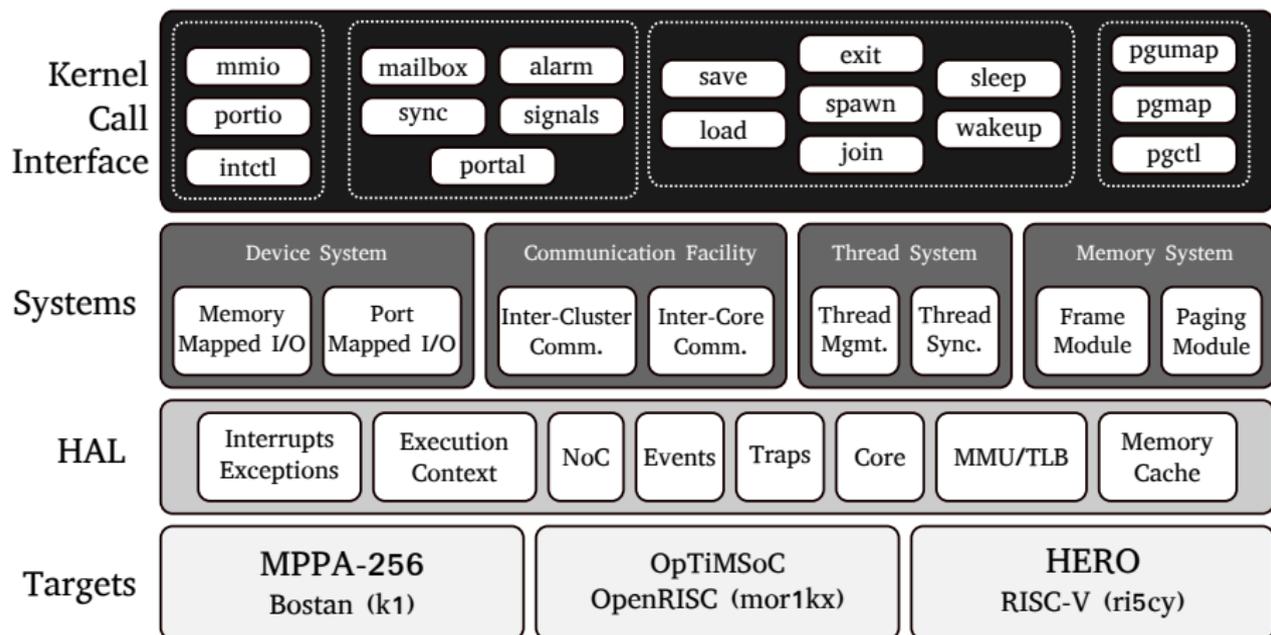


Figure: A detailed view of the Nanvix kernel.