High Performance Computing Systems

- Multikernels

Doug Shook

Multikernels

- Two predominant approaches to OS:
  - Full weight kernel
  - Lightweight kernel
- Why not both?
  - How does implementation affect usage and performance?
- Gerolfi, et. al. "A Multi-Kernel Survey for High-Performance Computing," 2016

FusedOS

- Assumes heterogeneous architecture
  - Linux on full cores
  - LWK requests resources from linux to run programs
- Uses CNK as its LWK

IHK/McKernel

- Uses an Interface for Heterogeneous Kernels
  - Resource allocation
  - Communication
- McKernel is the LWK
  - Only operable with IHK
- Uses proxy processes

mOS

- Embeds LWK into the Linux kernel
  - LWK is visible to Linux just like any other process
- Resource allocation is performed by sysadmin/user

FFMK

- Uses the L4 microkernel
  - What is a microkernel?
- Also uses a para-virtualized Linux instance
  - What is paravirtualization?

Hobbes

- Pisces Node Manager
- Kitten LWK
- Palacios Virtual Machine Monitor

Sysadmin Criteria

- Is the LWK standalone?
- Which kernel is booted by the BIOS?
- How and when are nodes partitioned?

Application Criteria

- What is the level of POSIX support in the LWK?
- What is the pseudo file system support?
- How does an application access Linux functionality?
- What is the system call overhead?
- Can LWK and Linux share memory?
- Can a single process span Linux and the LWK?
- Does the LWK support NUMA?

Linux Criteria

- Are LWK processes visible to standard tools like ps and top?
- Are modifications to the Linux kernel necessary?
- Do Linux kernel changes propagate to the LWK?

LWK Criteria

- How well is the LWK code isolated from Linux?
- How difficult is it for the LWK to track Linux changes?
- What is the cost of writing and maintaining the LWK?
- How large and complex is the LWK code?
- How much control does the LWK have over physical memory?
- What scheduling policy does the LWK provide?

Sysadmin Criteria

Project/ Property	FusedOS	IHK/ McKernel	mOS	Pisces/ Kitten	Kitten/ Palacios	FFMK (L4)	Criteria
Standalone LWK image	No	Yes	No	Yes	Yes	Yes	1.1
Node booted by	Linux	Linux	Linux	Linux	Kitten	L4	1.2
Resource partitioning	Static	Dynamic	Static	Dynamic	Dynamic	Dynamic	1.3
	Late	Late	Early	Late	Late	Late	

Application Criteria

Project/ Property	FusedOS	IHK/ McKernel	mOS	Pisces/ Kitten	Kitten/ Palacios	FFMK (L4)	Criteria
POSIX compatibility on LWK	Yes	Yes	Yes	No	No	Yes	2.1
/proc, /sys support on LWK	No	No	Yes	No	No	No	2.2
Access method to Linux features	Proxy	Proxy	Migrate	No	No	Migrate	2.3
Linux sys call overhead	High	High	High	-	-	High	2.4
Inter-kernel shared memory	Yes	Yes	Yes	Yes	Yes	Yes	2.5
Multi-kernel processes	No	No	No	No	No	Possible	2.6
NUMA support on LWK	No	No	Yes	No	No	No	2.7
Kernel level performance isolation	Yes	Yes	Yes	Yes	Yes	Yes	2.8

Linux Criteria

Project/ Property	FusedOS	IHK/ McKernel	mOS	Pisces/ Kitten	Kitten/ Palacios	FFMK (L4)	Criteria
Linux tools for LWK tasks	Yes	Yes LWK side	Yes Linux side	No -	No -	Yes Linux side	3.1
Unmodified Linux kernel	No	Yes	No	Yes	Yes	No	3.2
Linux updates propagate to LWK	No	No	Yes	No	No	No	3.3

LWK Criteria

Project/ Property	FusedOS	IHK/ McKernel	mOS	Pisces/ Kitten	Kitten/ Palacios	FFMK (L4)	Criteria
Isolated LWK code base	Yes	Yes	No	Yes	Yes	Yes	4.1
Impact of Linux changes	Minimal	Minimal	Code merge	Minimal	No	L4Linux port	4.2
Development effort	Small	Significant	Ideally small	Significant	Significant	Significant	4.3
Code size (kLOC)[2]	150	65	12	213 (entire Hobbes)		32	4.4
LWK control over physical memory	Full	Full	Full	Full	Total	Total	4.5
LWK memory type control	Full	Full	Full	Full	Total	Total	4.6
Virtual address space control	Total	Total	Linux	Total	Total	Linux	4.7
Scheduler in LWK	Cooperative	Cooperative	Cooperative	Cooperative	Cooperative	Time sharing	4.8
Device driver transparency	No	Yes	Yes	No	No	No	4.9

Conclusions

Unikernel

- Problem: HPC hardware is becoming more and more complex
  - Why?
- How does this affect software development?
  - The operating system?
- Solution: a unikernel?
- Lankes, et. al. "HermitCore—A Unikernel for Extreme Scale Computing," 2016

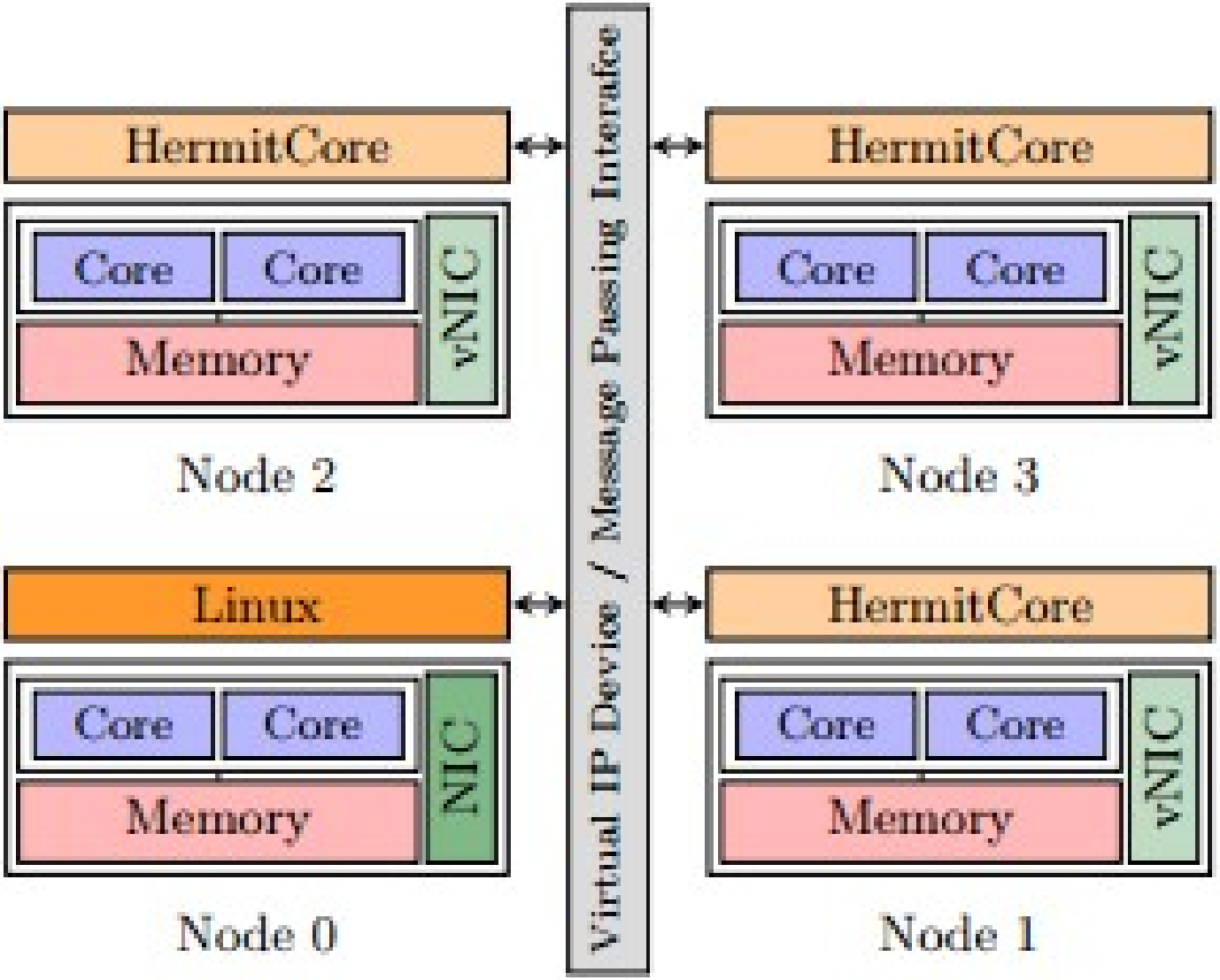
HermitCore

- OS scalability has three main approaches:
  - Stripped down OS
  - Developing a LWK from scratch
  - Multikernel
- HermitCore extends the multikernel approach
  - Instead of LWK uses a unikernel
- Focus: mapping of hardware to the software, rather than the OS.

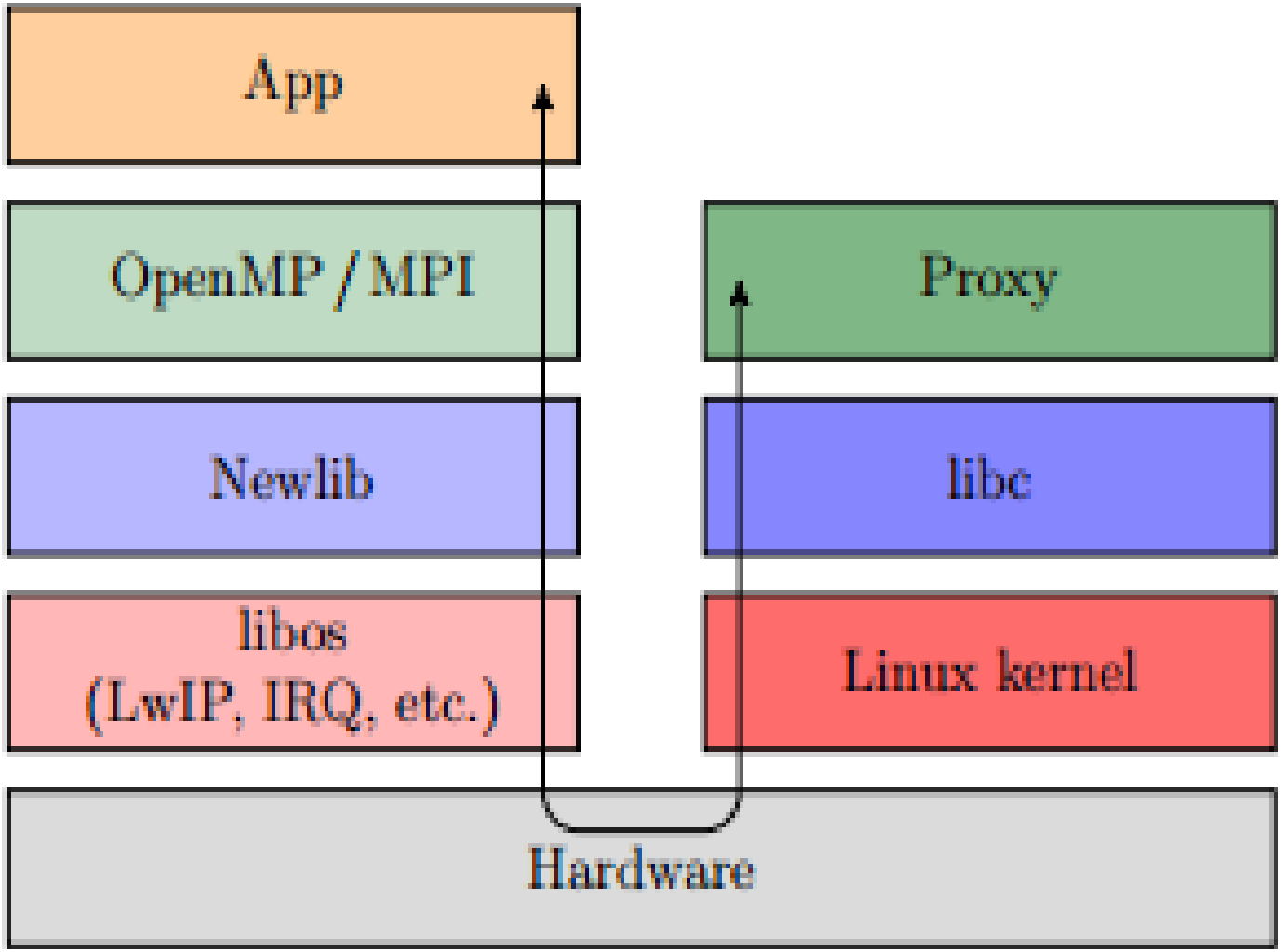
Design

- Main goals:
  - Reduction of OS noise
  - Predictable runtimes
  - Maintainability, extensibility, flexibility
  - Abstraction of hardware details
  - Support for common HPC programming models
  - Simple integration

Software Stack



Software Stack



Coexistence with Linux

- Utilizes hot plugging
- Proxy is responsible for registering HermitCore nodes with the OS
- Linux must do some memory management
- Only two files added to the OS

Toolchain

- Compiles using GCC
- OpenMP relies on pthreads
- Output is given in ELF format
- MPI is realized with the help of the RCCE library.

Performance

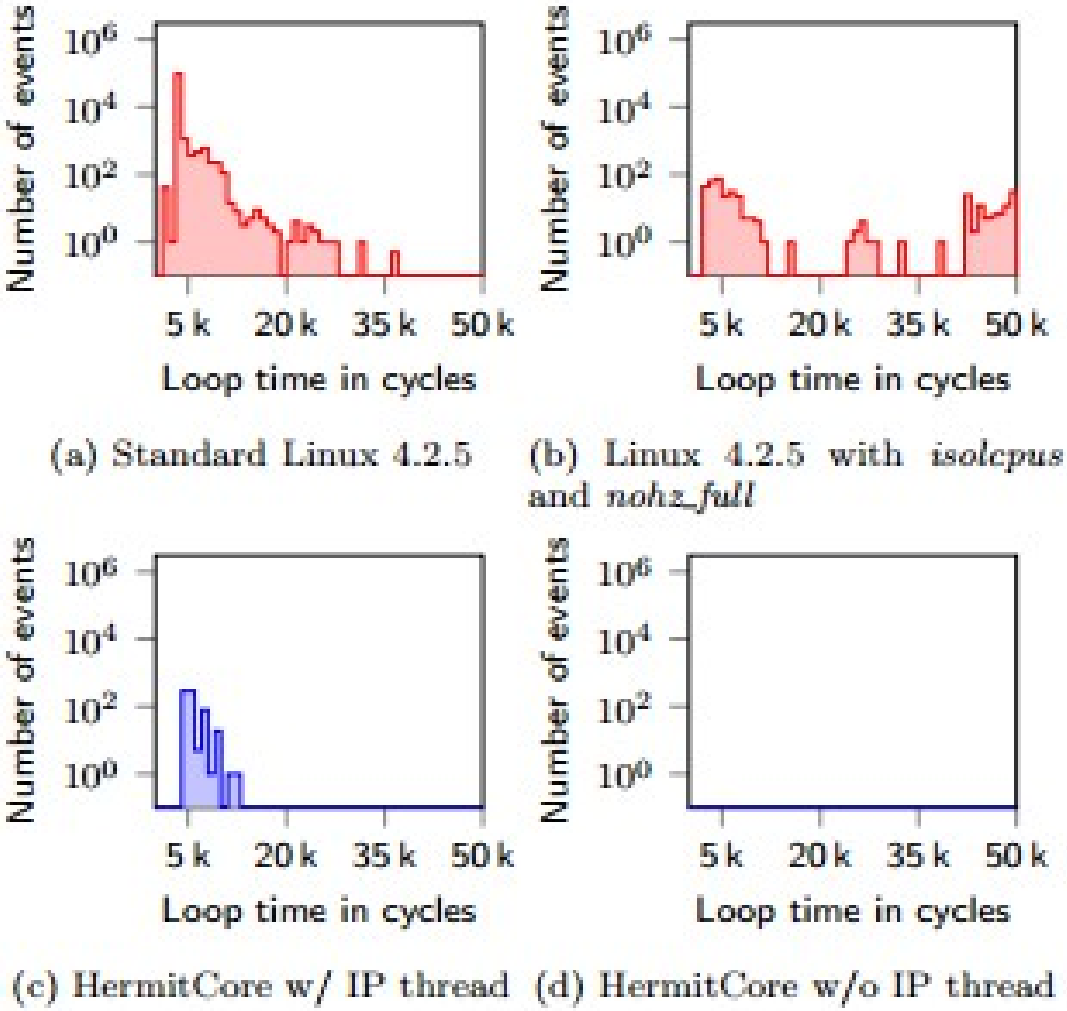
- 20 core system
  - 2.3GHz
  - 64GB DDR4
  - 25MB L3 Cache
- Compared against a traditional Fedora OS

System Call Overhead

System activity	HermitCore	Linux
`getpid()`	14	143
`sched_yield()`	97	370
`write()`	3520	1079
`malloc()`	3715	6575
first write access to a page	2018	4007

Hourglass Benchmark

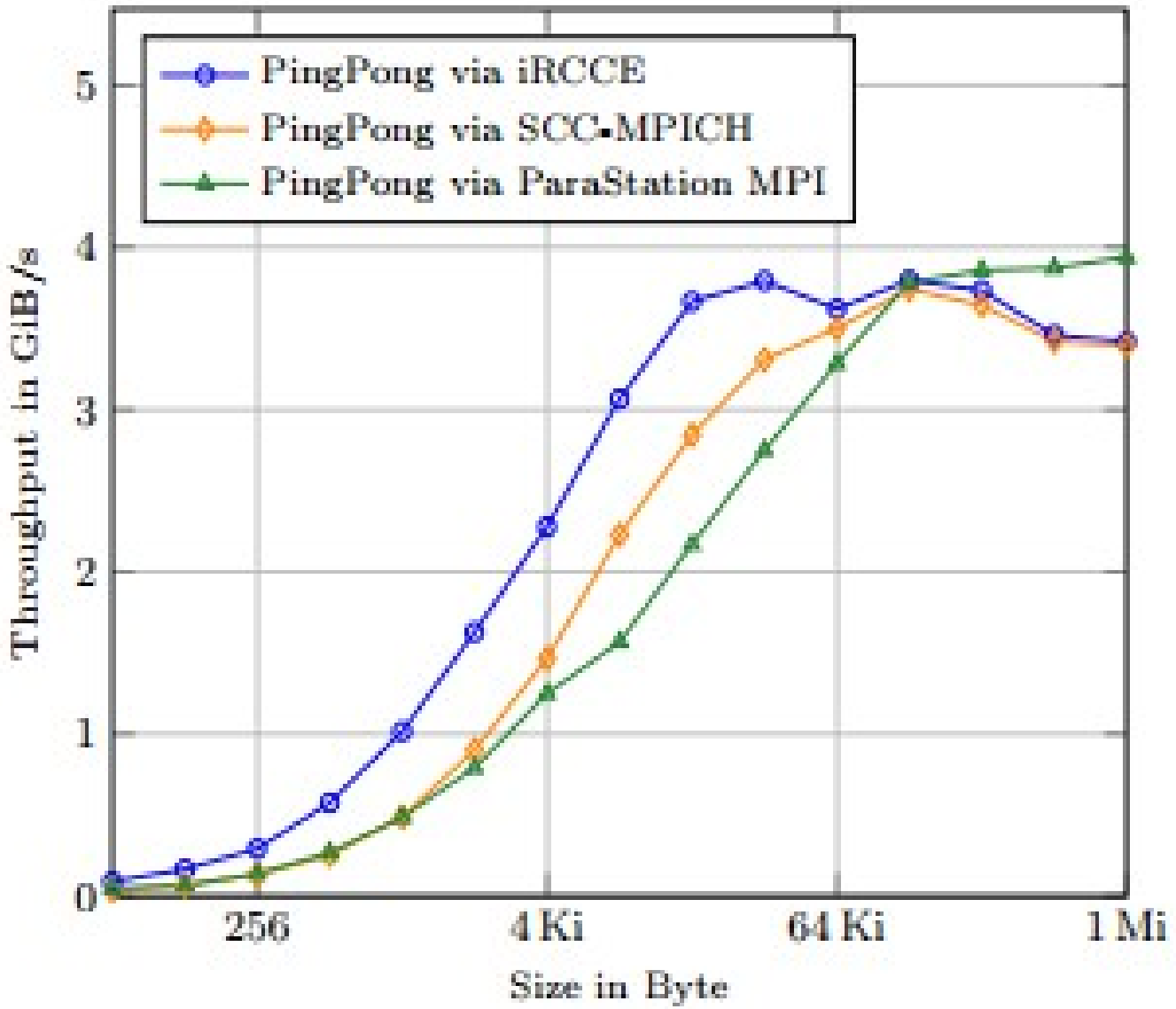
OS	Gaps	
	Avg	Max
Linux	69	31068
Linux (isolcpu)	69	51840
HermitCore (w/ LwIP)	68	12688
HermitCore (w/o LwIP)	68	376



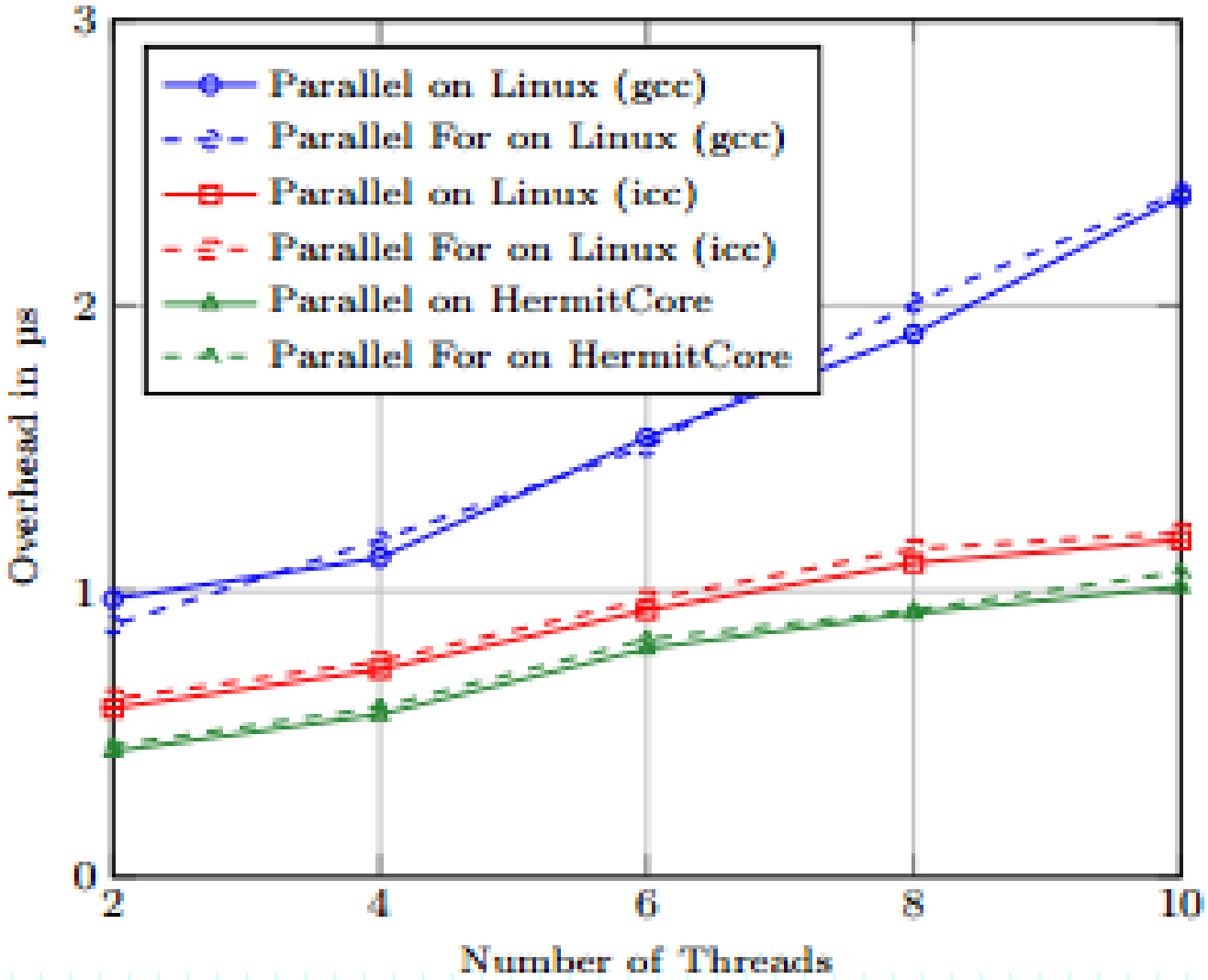
(a) Standard Linux 4.2.5 (b) Linux 4.2.5 with *isolcpus* and *nohz_full*

(c) HermitCore w/ IP thread (d) HermitCore w/o IP thread

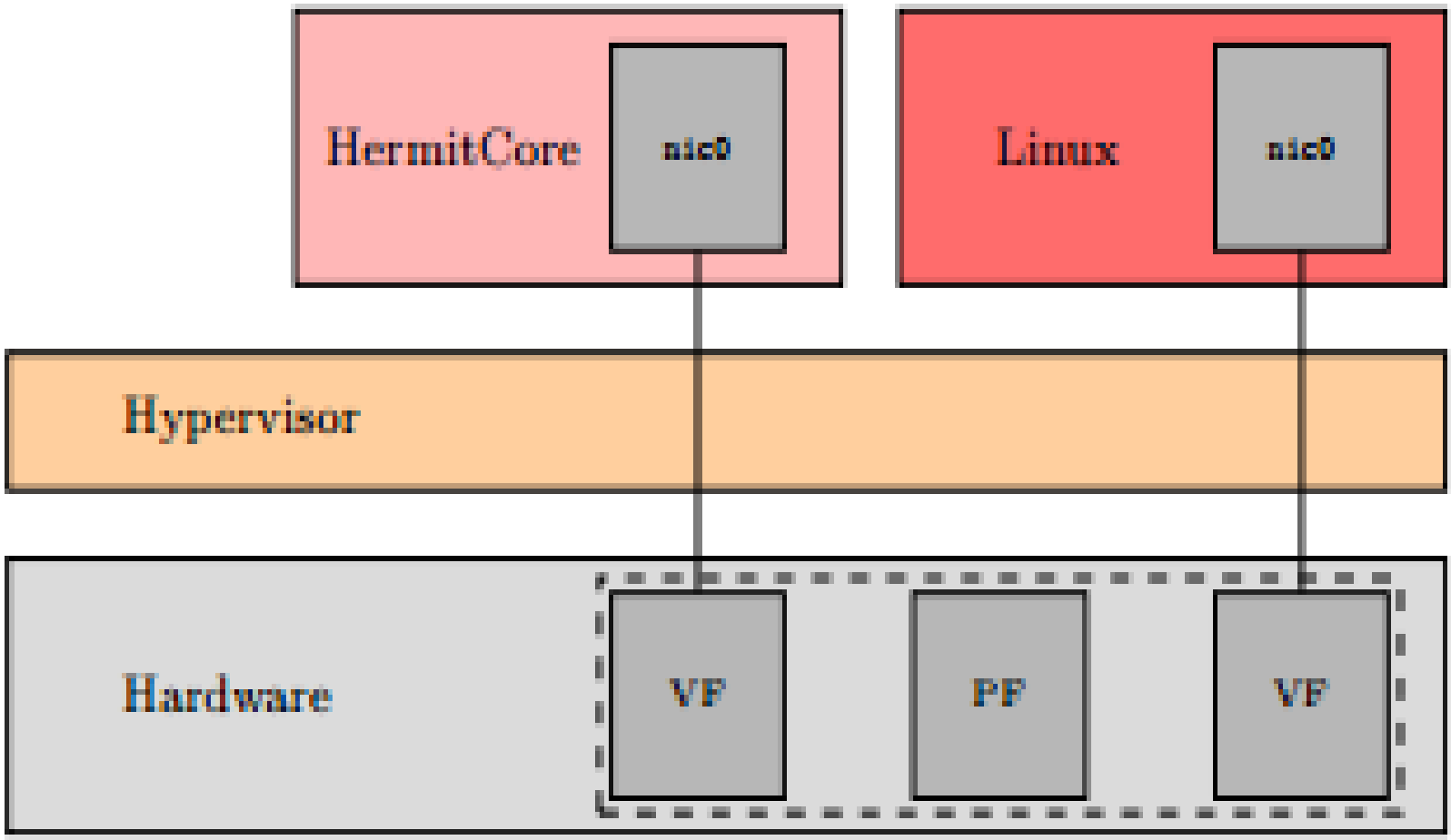
Inter-kernel Communication



OpenMP Synchronization



Future Work



Conclusion