When Slower is Faster: On Heterogeneous Multicores for Reliable Systems

Tomas Hruby, Herbert Bos and Andrew S. Tanenbaum
The Network Institute, VU University Amsterdam

Usenix ATC 2013

Outline

1. Introduction
2. NewtOS
3. Evaluation
4. Discussion

Outline

1. Introduction
2. NewtOS
3. Evaluation
4. Discussion

Motivation

- More and more vendors develop heterogeneous cores (ARM big.LITTLE, NVIDIA Tegra-3, Xeon Phi, ...).
- All cores share a large subset of the ISA, but have different. performance characteristics.
- There has been a lot of research about them, but mostly focused on applications and not on the operating system.
- They want to explore how these architectures can help to balance performance and resource consumption.

Contributions

1. They explore hardware design space of future platforms with current ones.
2. They show that performance is equal or better with slower cores.
3. The system has the potential for dynamic reconfiguration.

Outline

1 Introduction

2 NewtOS

3 Evaluation

4 Discussion

What is it?

- High-performance clone of MINIX 3.
- Has the same reliability.
- Even core OS components can be replaced on the fly.
- If components crash, they can restart them.
- Often that's transparent for applications.

Design of the network stack

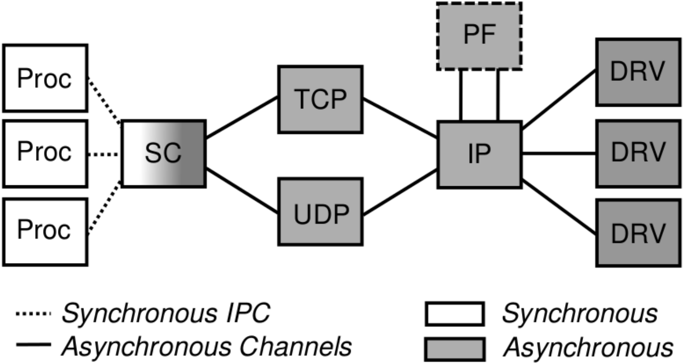


Figure 1: Design of the NewtOS network stack

Dynamic reconfiguration

- Each system component can run on a dedicated core.
- Or it can share a core with others.
- If the workload changes, the system can redistribute itself.

Non-overlapping ISA

- They claim that NewtOS' live update functionality can be used for migration.
- This can be done by recompiling the same code and replacing it while it's running.
- Is done only at the top of the main loop.
- If memory layout changes, a transition function is generated.
- Can of course also be done by precompiling an application for multiple ISAs.

Outline

1 Introduction

2 NewtOS

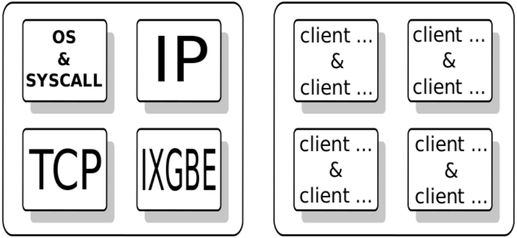
3 Evaluation

4 Discussion

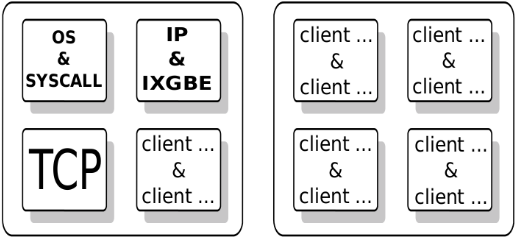
Introduction

- Done with dual socket quad-core Intel Xeon E5520.
- Two ways to scale the performance of a core:
  1. Scale frequency of a whole chip down (2.3Ghz .. 1.6Ghz)
  2. Thermal throttling per core (in steps of 12.5% of clock speed)
- They use frequency scaling from 2.3Ghz to 1.6Ghz and thermal throttling from 1.6Ghz to 0.2Ghz.
- Benchmarks are done by running iperf on a Linux machine and connecting to it from NewtOS (can achieve 10G when using Linux to connect).
- CPU utilization is measured by the time spent in userspace (started/ stopped before and after a kernel call)

Test configurations



(a) Configuration #1 - dedicated cores



(b) Configuration #2 - hyper-threading

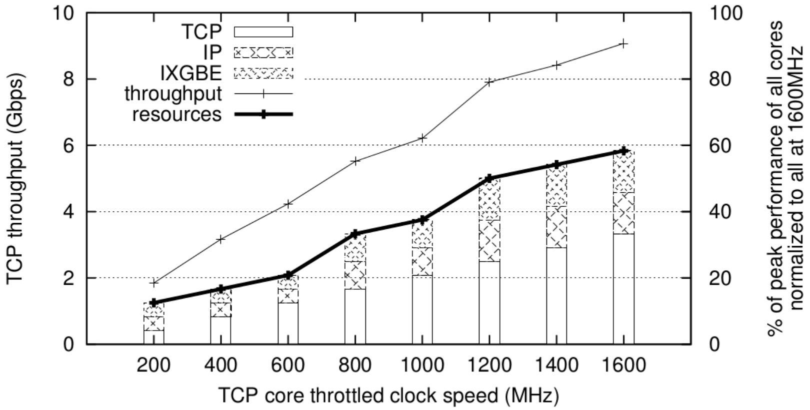
Frequency scaling (2.3Ghz - 1.6Ghz)

MHz	drop	Mbps	drop	TDP(W)	drop
2267	–	8641	–	80	–
1867	18%	8152	6%	48	40%
1600	30%	7840	9%	34	57%

Table 3: Performance loss versus potentially saved power

Important: TCP uses the core to 70%, IP and driver to 40%.

Thermal throttling (1.6Ghz - 0.2Ghz)



Thermal throttling: CPU utilization

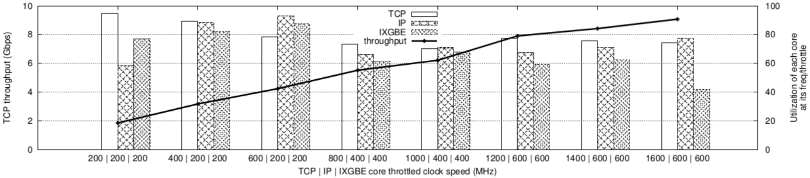


Figure 4: Configuration #1 – CPU utilization of each core throttled to % of 1600 MHz.

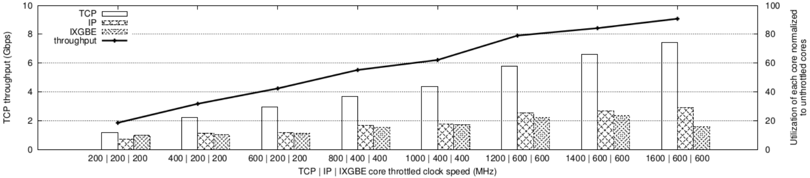
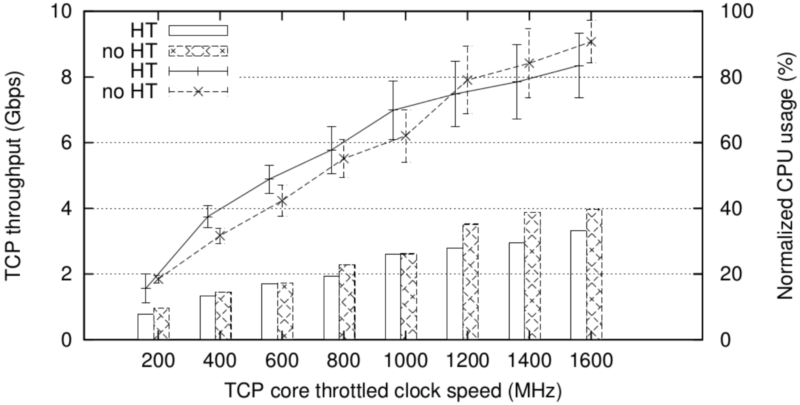
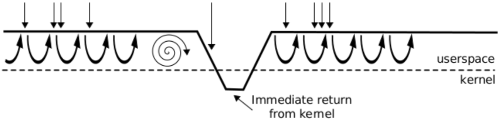


Figure 5: Configuration #1 – CPU utilization normalized to cores at 1600 MHz unthrottled.

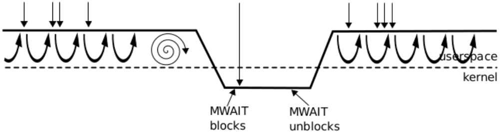
Hyperthreading



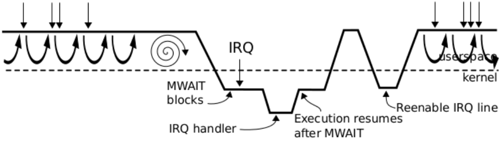
Why is slower faster?



(a) Job arrives between deciding to sleep and halting the core



(b) Job arrives when execution is suspended on MWAIT



(c) Driver - an IRQ arrives when execution is suspended

Outline

1 Introduction

2 NewtOS

3 Evaluation

4 Discussion

Major point of criticism

- They don't really explain what is happening there.
- This is especially bad because the claim "slower is faster" is really bold.
- My guess is: if IP and the driver have low CPU utilization, they sleep often. This introduces latency which prevents that TCP can use the CPU more than 70-75%.
- If that is the case, I'm wondering why...
  - they stop to throttle IP and driver up at 600 Mhz instead of showing what happens if they go further to 1600Mhz,
  - they haven't shown what happens if they poll and
  - they don't say that this is the bottleneck and the reason why scaling the chip down decreases the throughput although TCP doesn't use the CPU to 100%.

Other points

- Using 2 ways of slowing down the cores makes it difficult to draw a connection between them. Why don't they only use throttling?
- They mention power consumption, but don't really evaluate it, although this is a major point of the paper (using less resources while reaching the same performance).
- Doesn't the whole approach conflict with flow-control? Both try to scale the speed until it fits.
- Good point: The idea of scaling down cores (or assigning applications to slower cores) to reach a high utilization instead of sleeping is quite nice (if you can give an application its own core).