Rethinking Network Performance

Factoring In Power

Netdev conf 0x17, Vancouver

Nabil Bitar, Jamal Hadi Salim, Pedro Tammela

Motivation

- Power and silicon costs are rising
  - Begs the question: Should we upgrade this rack? Can we extend it's life somehow?
- Green Networking
- Chip vendors are deploying P/E Cores across their products
  - Trade some performance for better Energy efficiency
- Offload is hitting mainstream for many applications
  - PCI devices are getting more and more features
- AI, AI, AI, AI…
  - Scale of deployment massive

Sources of Power consumption

- CPU
- Fans
- DRAM
- PCI devices
  - Note:Some NICs and GPUs draw extra power outside of the PCI bus
- Disks
- Coprocessors (ex: QAT)
- Silicon
- And possibly others…

Rethinking Performance

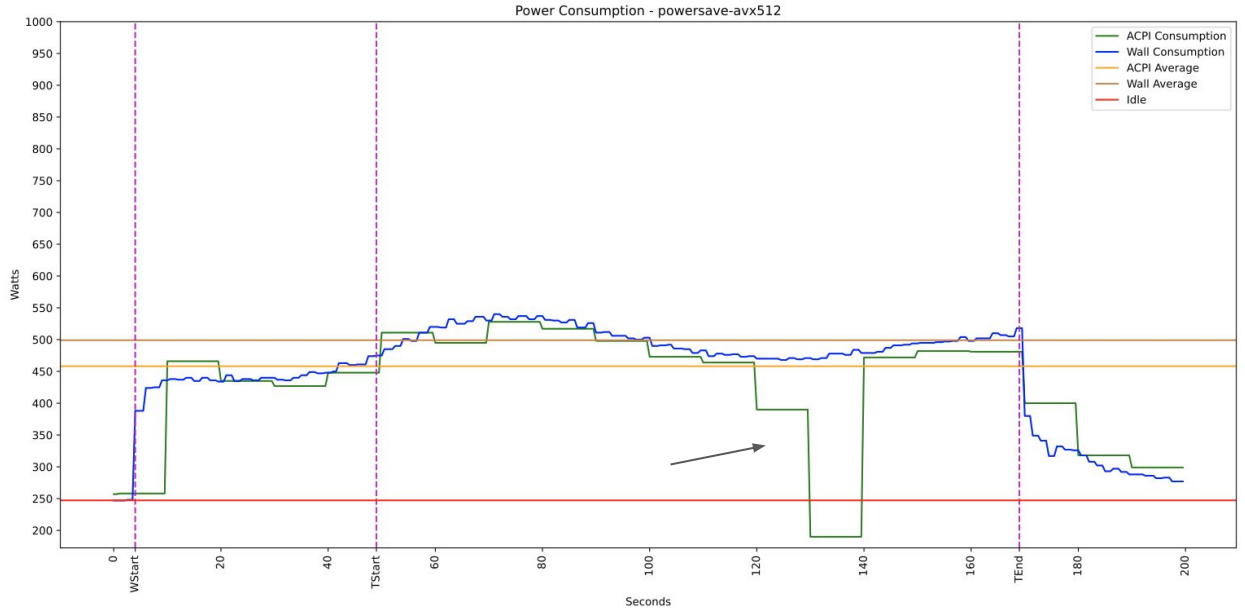
- Not all processors are the same
- Transform “10 Mpps per core” mentality into “10Mpps per watt”

Measuring Power

Measuring Power Consumption

- ACPI Sensors
  - lm-sensors
  - ACPId
  - Powertop
- External power measurements devices
  - Power PDU
- BMC
  - Usually sensor reading
- Proprietary vendor tools (more accurate)
  - Vendor CPU tooling (eg Intel PTAT)
  - NIC specific
  - PCI hardware tools

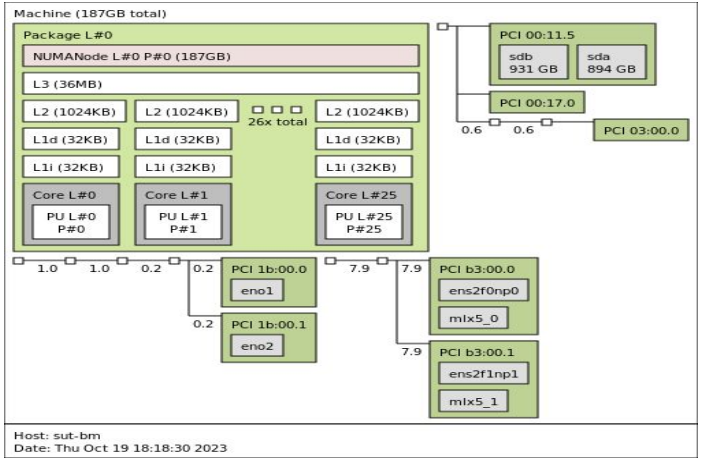
ACPI vs External PDU

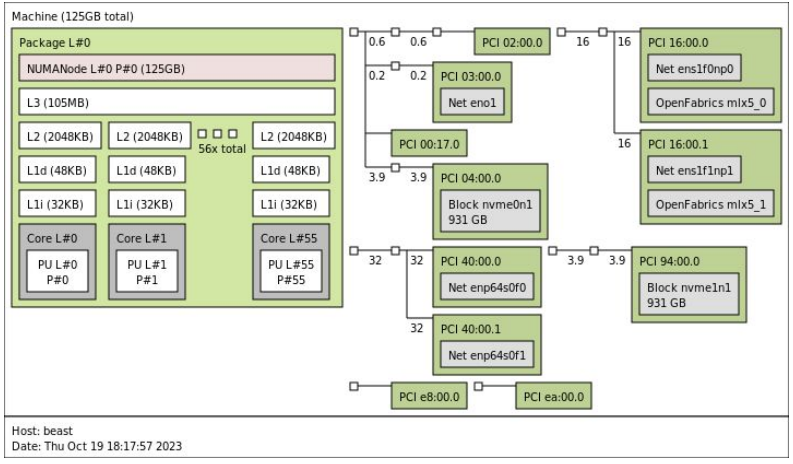


Test Setup

Server/SUT Specification

- Server machine (HT off, constrained to 4 Cores):
  - Server1, 2023 (CPU: 56C/112HT))
    - 128GB RAM DDR5 4800Mhz
  - Server2, 2020 (26C/52HT) - notice we have a single core
    - 192G RAM
  - 2x25 Gbps Ethernet link
  - Bluefield 2 (2x25G)





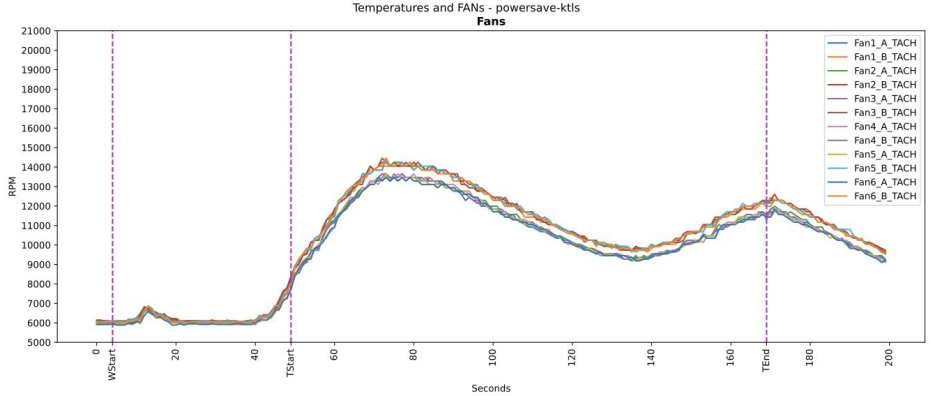
Measurement Setup

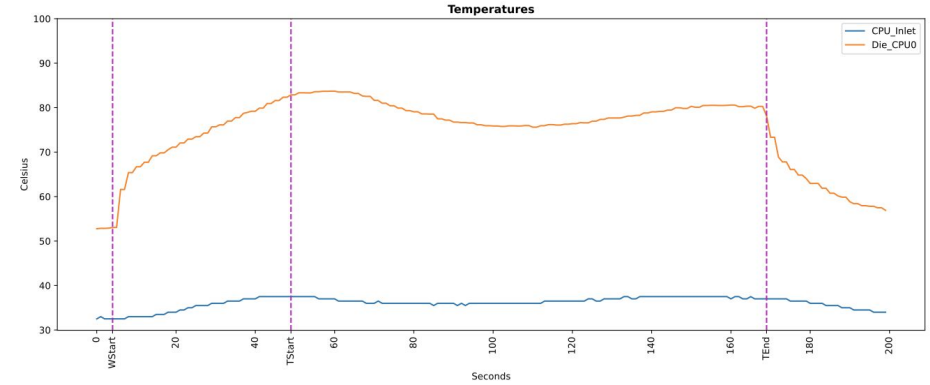
- Connect server and client to PDU
  - Power connectors C13/14
  - Restful endpoint
    - Power info extraction
      - 99.99% accuracy

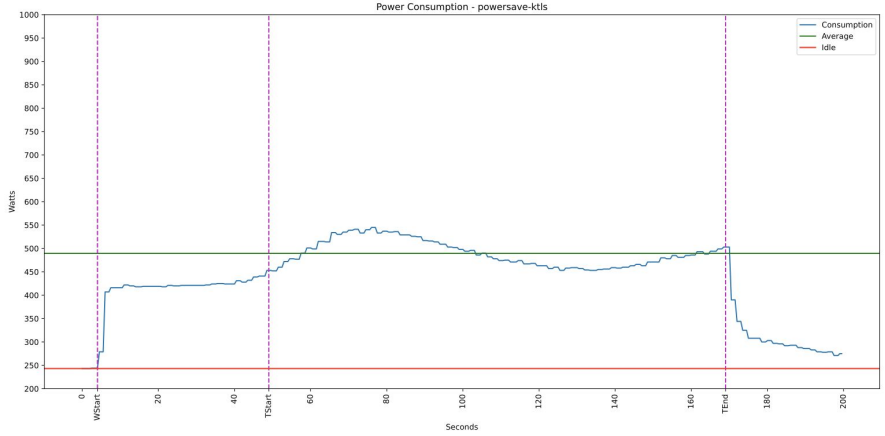
- Server1 side has BMC and exposes Redfish http restful endpoint
  - Fans and thermal info extraction
- Server2 has BMC but requires license
  - But can control fans via IPMI

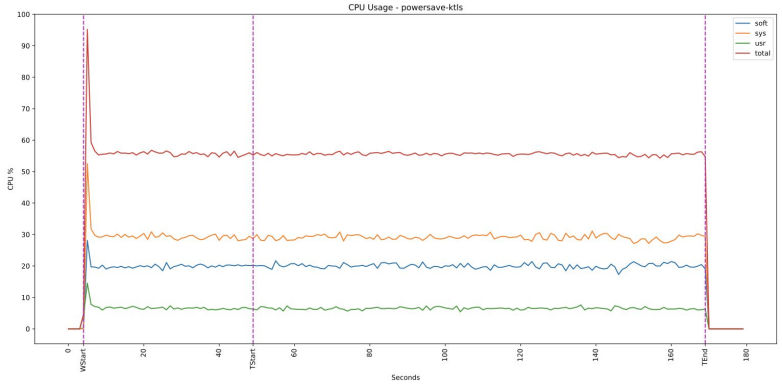
Summary Results

Example Measurement: kTLS (server 1) - 50G 16K





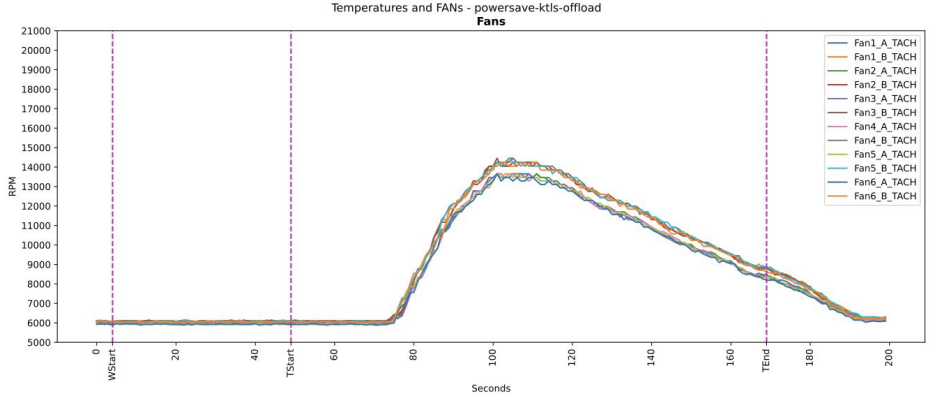


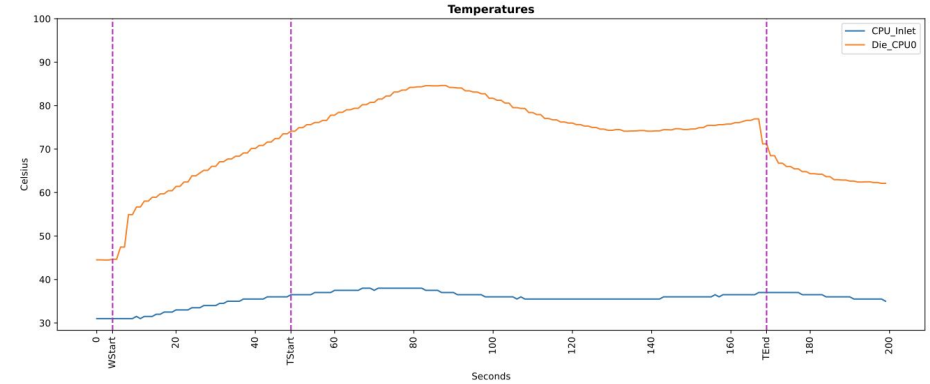


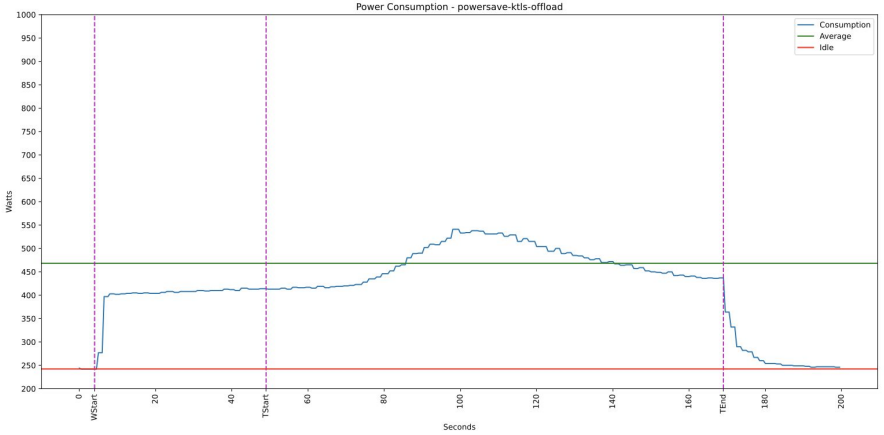
Perf: Software kTLS (server 1) - 50G 16K

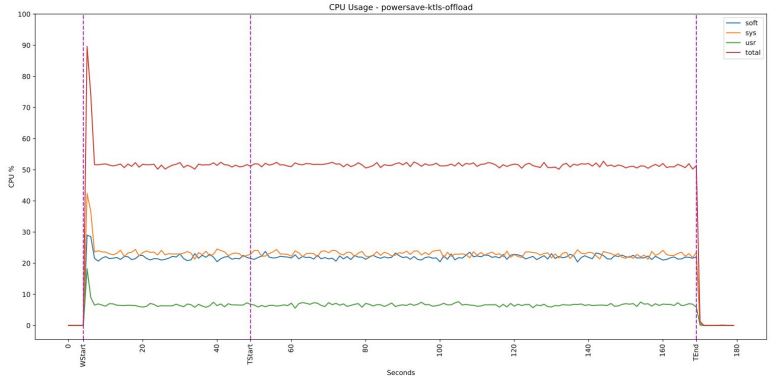
```
  8.03%  [kernel]          [k] _encrypt_by_8_new17529
  5.05%  [kernel]          [k] clear_page_erms
  4.23%  [kernel]          [k] tasklet_action_common.constprop.0
  2.58%  [kernel]          [k] _raw_spin_lock
  1.91%  [kernel]          [k] skb_release_data
  1.83%  [kernel]          [k] __alloc_skb
  1.38%  [kernel]          [k] mlx5_eq_comp_int
  1.06%  [kernel]          [k] sk_msg_alloc
  1.04%  [kernel]          [k] native_irq_return_iret
  1.01%  [kernel]          [k] crypto_stats_aead_encrypt
  0.97%  [kernel]          [k] _raw_spin_lock_irqsave
  0.82%  [kernel]          [k] crypto_stats_get
  0.78%  [kernel]          [k] memcpy_erms
  0.76%  [kernel]          [k] slab_free_freelist_hook.constprop.0
  0.70%  [kernel]          [k] read_tsc
  0.67%  [kernel]          [k] filemap_get_read_batch
  0.67%  nginx             [.] ngx_open_cached_file
  0.63%  [kernel]          [k] __inet_lookup_established
  0.62%  [kernel]          [k] tcp_v4_rcv
  0.62%  [kernel]          [k] kmem_cache_free
  0.61%  [kernel]          [k] tls_sw_do_sendpage
  0.60%  [kernel]          [k] tcp_clean_rtx_queue.constprop.0
  0.59%  [kernel]          [k] page_frag_free
  0.59%  [kernel]          [k] __slab_free
  0.59%  [kernel]          [k] crypto_stats_aead_decrypt
  0.58%  [kernel]          [k] _raw_read_lock_irqsave
  0.57%  [kernel]          [k] __tcp_transmit_skb
  0.57%  [kernel]          [k] rb_first
```

kTLS Offload (server 1)- 50G 16K









kTLS Offload (server 1) - 50G 16K

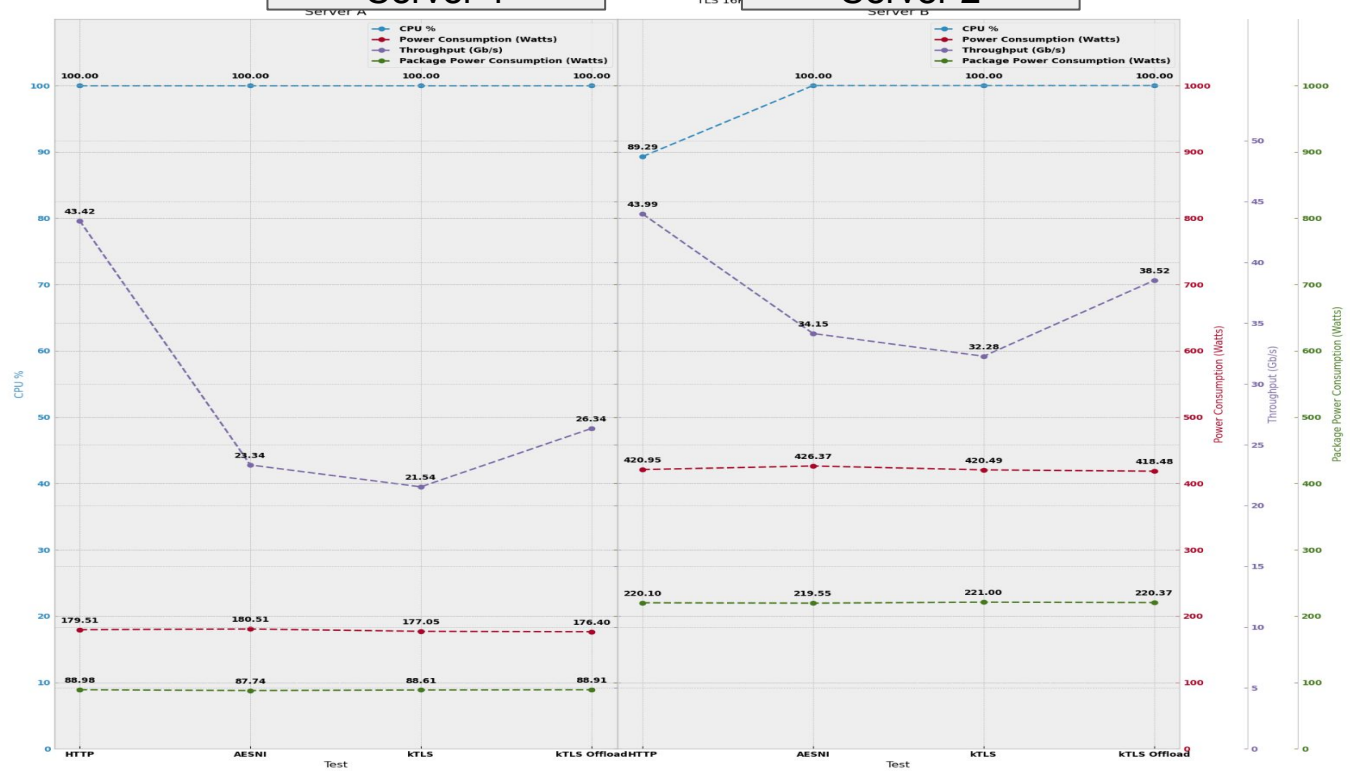
```
  5.25%  [kernel]           [k] clear_page_erms
  4.36%  [kernel]           [k] tasklet_action_common.constprop.0
  2.71%  [kernel]           [k] _raw_spin_lock
  2.43%  [kernel]           [k] __alloc_skb
  2.33%  [kernel]           [k] memcpy_erms
  1.60%  [kernel]           [k] skb_release_data
  1.58%  [kernel]           [k] slab_free_freelist_hook.constprop.0
  1.56%  [kernel]           [k] _raw_spin_lock_irqsave
  1.44%  [kernel]           [k] mlx5_eq_comp_int
  1.25%  [kernel]           [k] tls_push_record
  1.08%  [kernel]           [k] native_irq_return_iret
  0.93%  [kernel]           [k] filemap_get_read_batch
  0.92%  [kernel]           [k] destroy_record
  0.88%  [kernel]           [k] page_frag_free
  0.86%  [kernel]           [k] tcp_v4_rcv
  0.86%  [kernel]           [k] tls_push_data
  0.85%  [kernel]           [k] read_tsc
  0.82%  [kernel]           [k] tcp_clean_rtx_queue.constprop.0
  0.76%  [kernel]           [k] kfree
  0.75%  [kernel]           [k] _raw_read_lock_irqsave
  0.75%  [kernel]           [k] __mod_timer
  0.71%  [kernel]           [k] __slab_free
  0.70%  [kernel]           [k] __inet_lookup_established
  0.65%  [kernel]           [k] kmem_cache_free
  0.65%  [kernel]           [k] tcp_rcv_established
  0.62%  [kernel]           [k] tls_icsk_clean_acked
  0.62%  [kernel]           [k] tcp_trim_head
  0.60%  [kernel]           [k] tls_device_decrypted
```

Varying TLS approaches

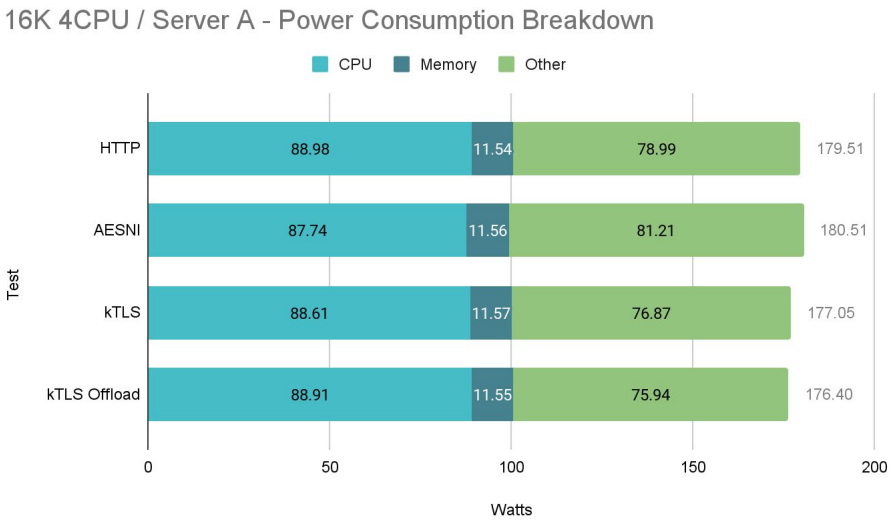
TLS 16K - 4CPUs

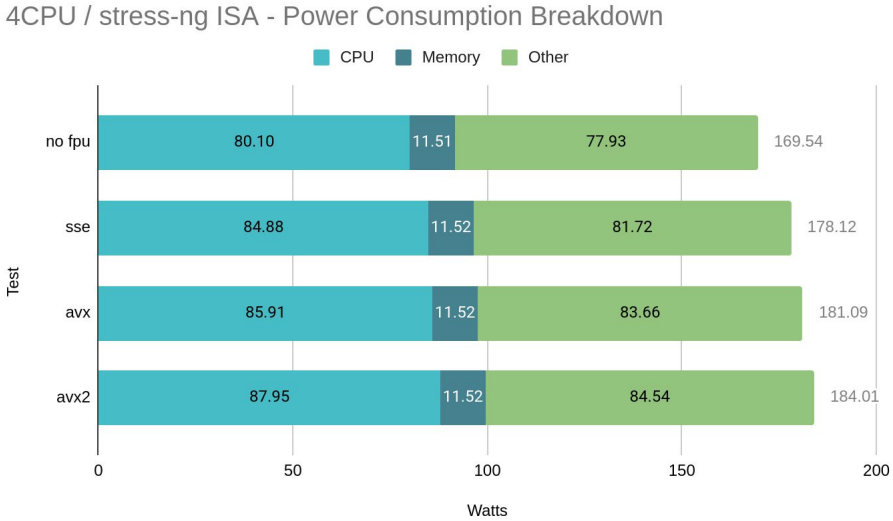
Server 1

Server 2

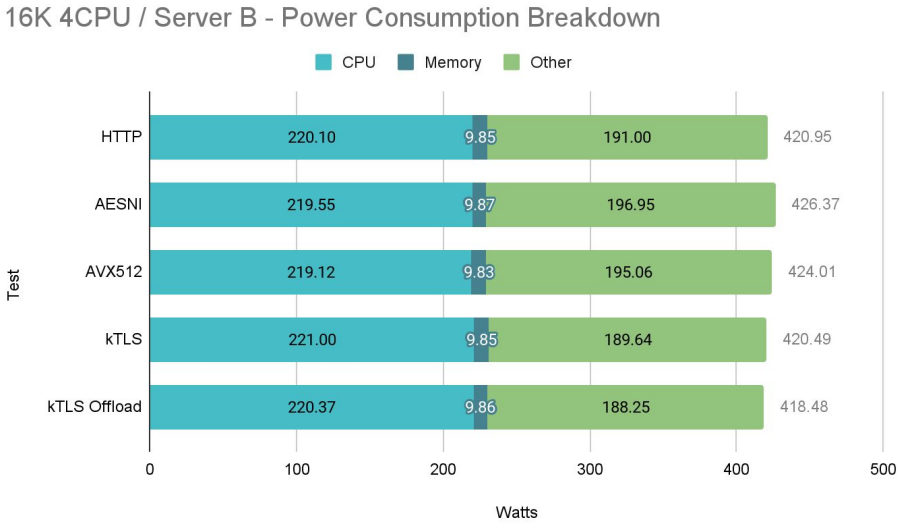


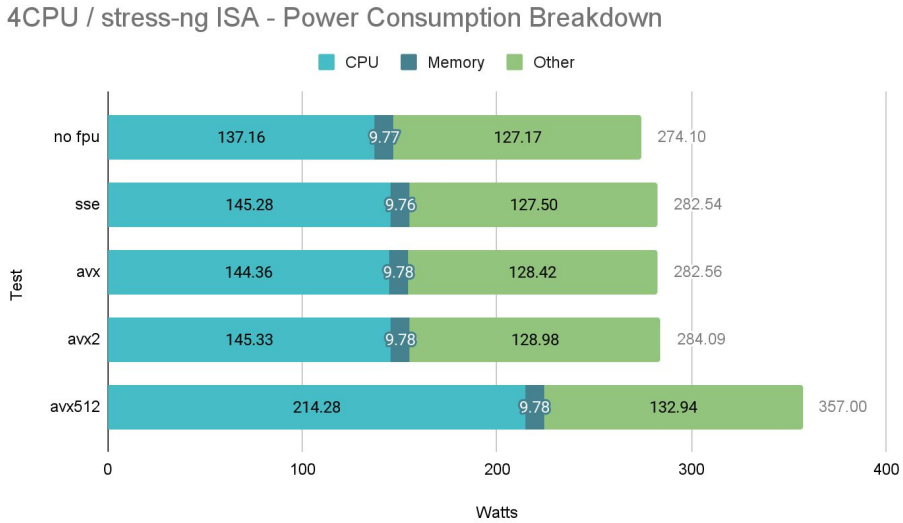
Server 1 - Power consumption breakdown (PTAT)



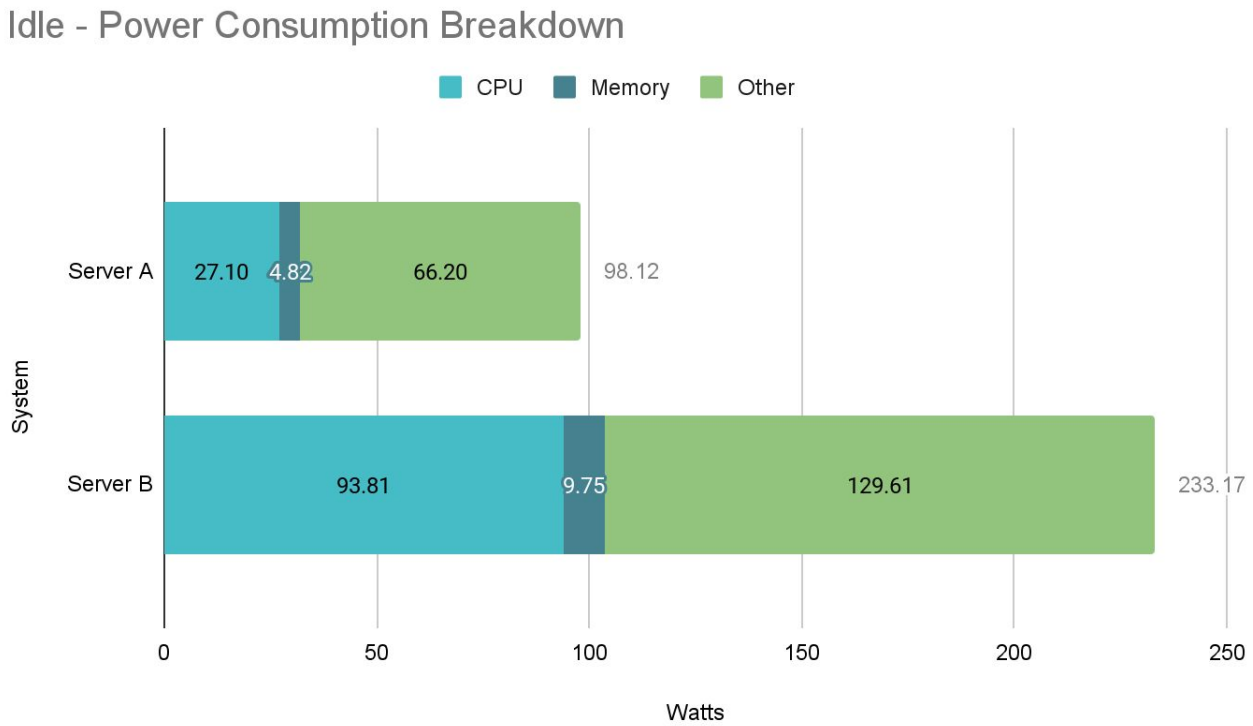


Server 2- Power consumption breakdown (PTAT)





Idle - Power consumption breakdown (PTAT)



Conclusions

- System level power consumption is a complex metric
  - Every peripheral in the system contributes to it
  - Hard to reproduce without identical rack
    - Even then the ambient temperature could be a factor
  - Thermal solutions vary widely in the field
- Wall plug measurements are super valuable
  - Don’t miss out on any Watt consumed/wasted!
  - Evaluate the rack as a whole

Back Slides

Setup For NGINX Tests

- Powersaving mode
  - 800Mhz - 3.8Ghz
  - EPB value of 15 (high energy savings)
- Performance mode
  - 3.8 Ghz
  - EPB value of 0

EPB value	String
0	performance
4	balance-performance
6	normal, default
8	balance-power
15	power

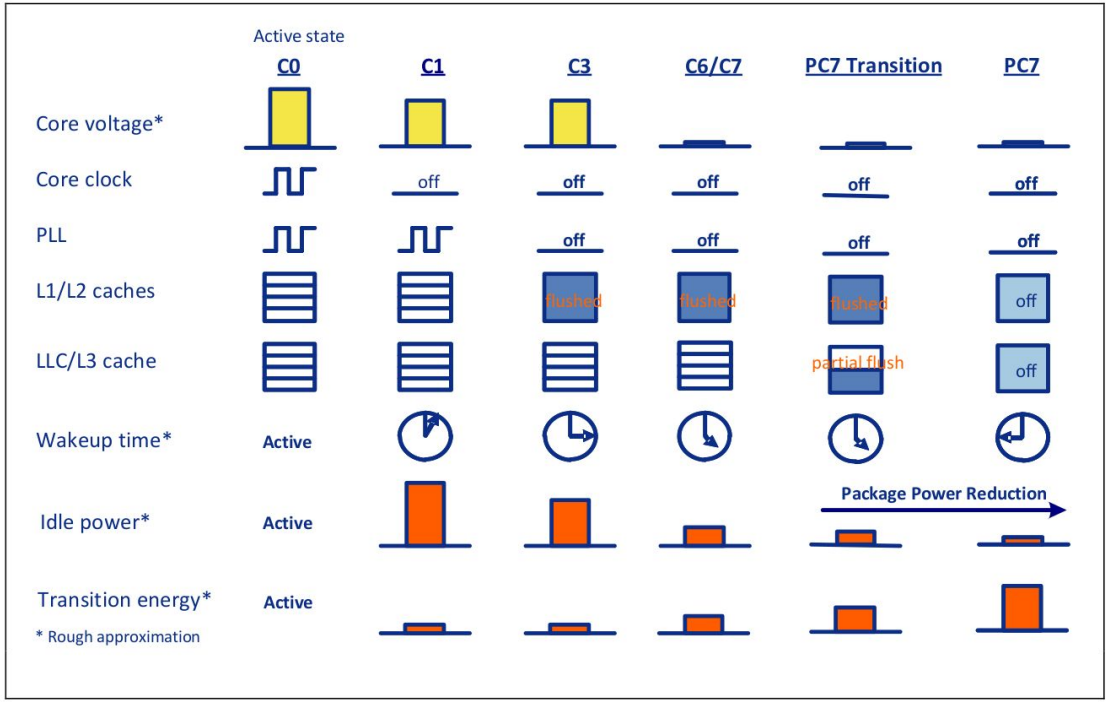
TDP and Turbo Frequency

- TDP acts as a ceiling of how much total power the CPU draw
- With Turbo frequency, TDP also works as an “attractor” point
  - The CPU evaluates the workload and adjusts the frequencies (Core and Uncore) up or down in order to reach the CPU’s TDP
  - That means workloads consuming few cores can draw power close to TDP
- Turbo frequencies get lower as more and more cores are used
  - Depending on the application it can push the frequencies further down
  - In terms of cost: SSE < AVX2 < AVX512
- Disabling turbo frequency makes the CPU consume less power
  - On Sapphire Rapid we saw a 50W reduction in the CPU Package consumption
  - Caps the frequencies possibly missing out on performance

Intel PState vs CState

- CStates and PStates are used for power management of the CPU Package
  - CStates controls the sleep states
  - PStates controls the performance states
- PStates were initially managed by the OS. Now they are controlled by the CPU itself
  - Intel SpeedShift Technology
- Every CPU executes tasks on CState 0
  - From there PStates will kick in as well as Turbo Boost
- Transitions from higher CStates to CState 0 are known to cause latency increase
- PStates can be throttled if temperatures are not controlled

Intel PState vs CState



TDP and Turbo Frequency

- TDP acts as a ceiling of how much total power the CPU draw
- With Turbo frequency, TDP also works as an “attractor” point
  - The CPU evaluates the workload and adjusts the frequencies (Core and Uncore) up or down in order to reach the CPU’s TDP
  - That means workloads consuming few cores can draw power close to TDP
- Turbo frequencies get lower as more and more cores are used
  - Depending on the application it can push the frequencies further down
  - In terms of cost: SSE < AVX2 < AVX512
- Disabling turbo frequency makes the CPU consume less power
  - On Sapphire Rapid we saw a 50W reduction in the CPU Package consumption
  - Caps the frequencies possibly missing out on performance

Intel PState vs CState

- CStates and PStates are used for power management of the CPU Package
  - CStates controls the sleep states
  - PStates controls the performance states
- PStates were initially managed by the OS. Now they are controlled by the CPU itself
  - Intel SpeedShift Technology
- Every CPU executes tasks on CState 0
  - From there PStates will kick in as well as Turbo Boost
- Transitions from higher CStates to CState 0 are known to cause latency increase
- PStates can be throttled if temperatures are not controlled