Parallelization and non-functional properties leading to the concept of real-power

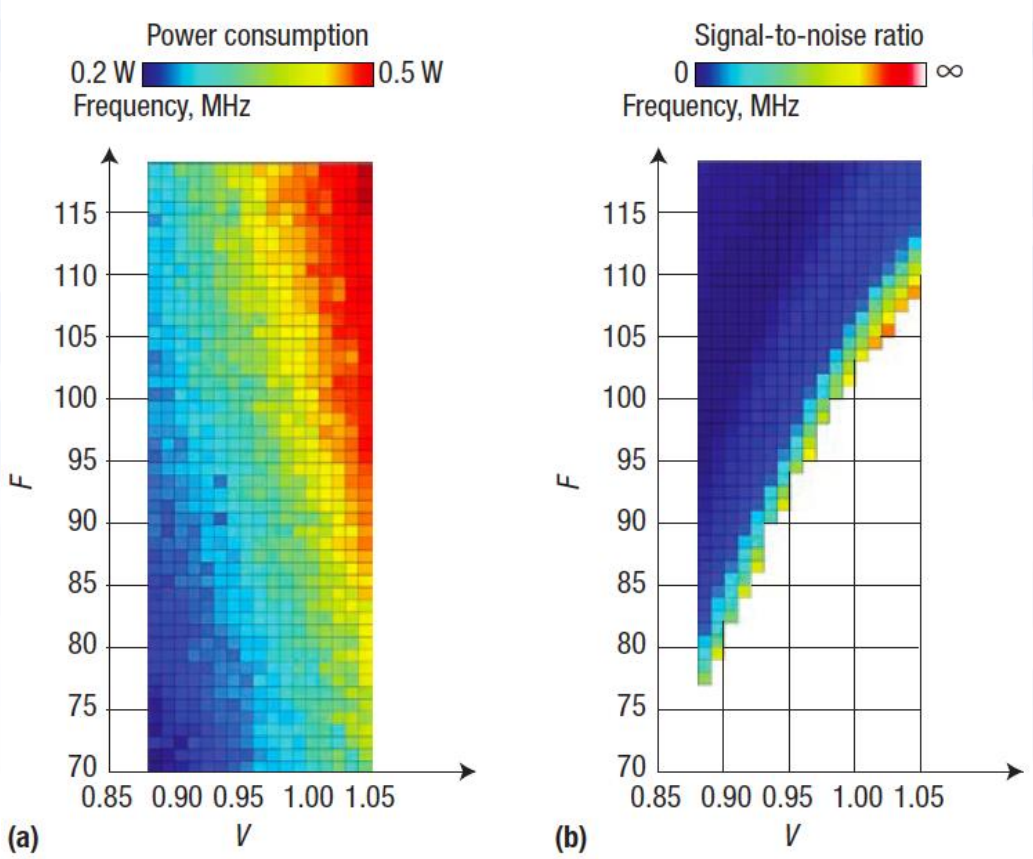
PRiME Project Newcastle Team (Fei Xia)
Schools of Engineering and Computing, Newcastle University

Non-functional parameters

- Performance (P)
- Energy (E)
- Reliability (R)
- How do PER trade off with each other?

F. Xia et al., "Voltage, Throughput, Power, Reliability, and Multicore Scaling," *Computer*, vol. 50, no. 8, pp. 34-45, 2017.

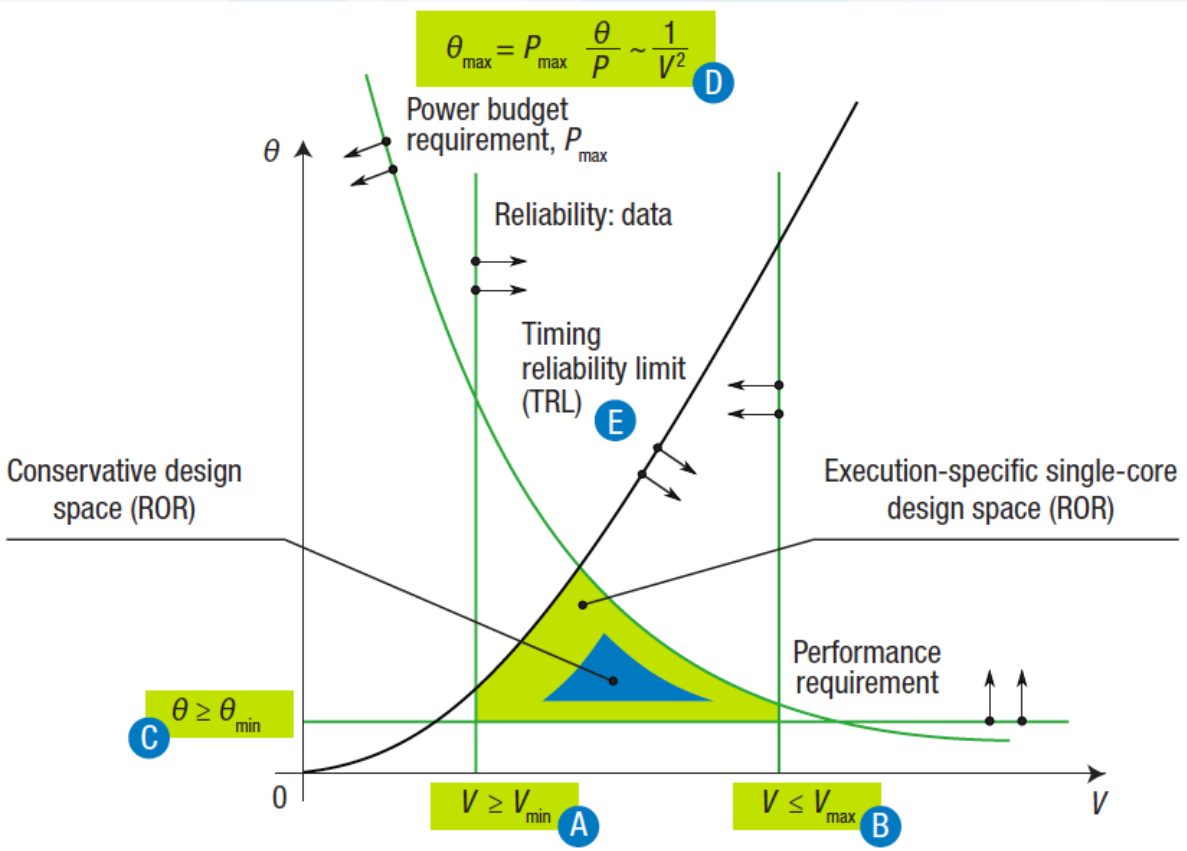
A real example



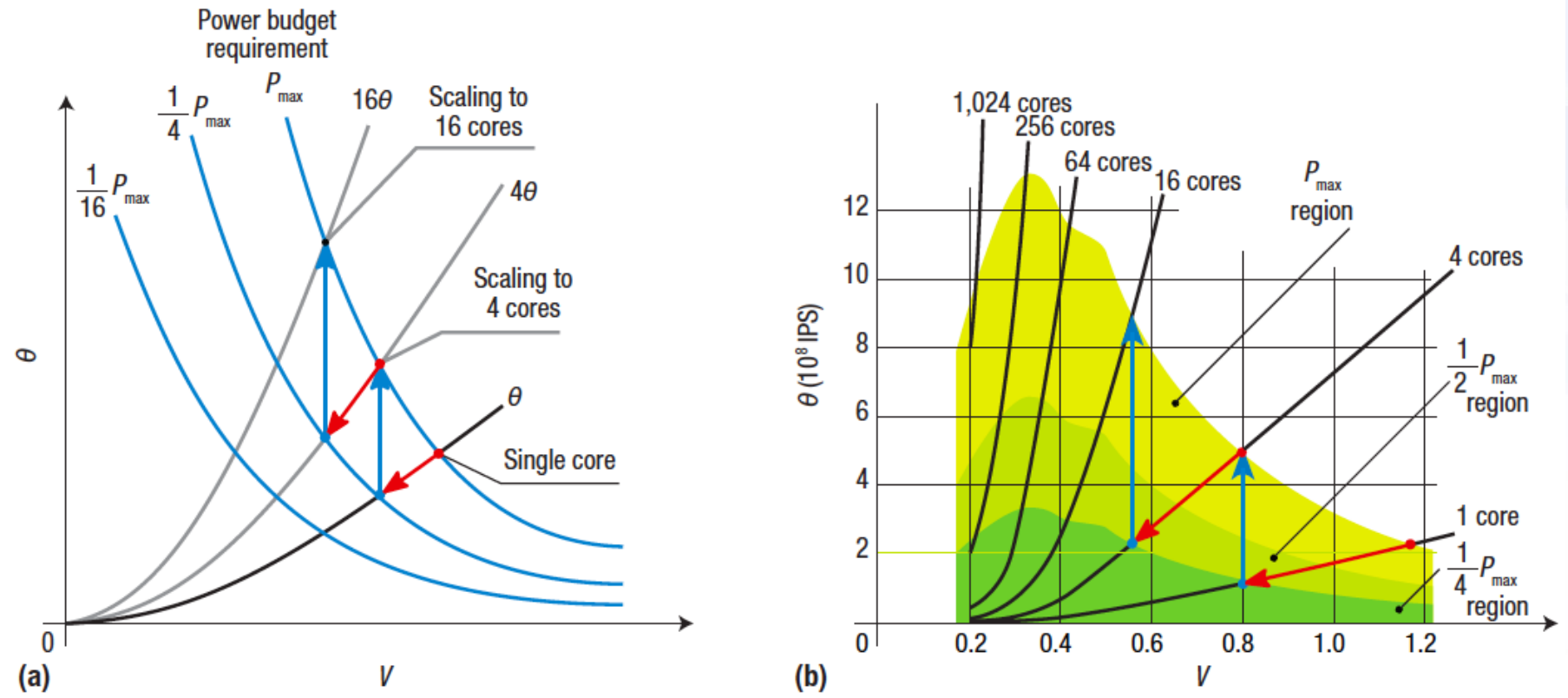
PER tradeoff

Performance
Energy
Reliability (PER)

Reliable
Operating
Region (ROR)



Scaling to multiple cores

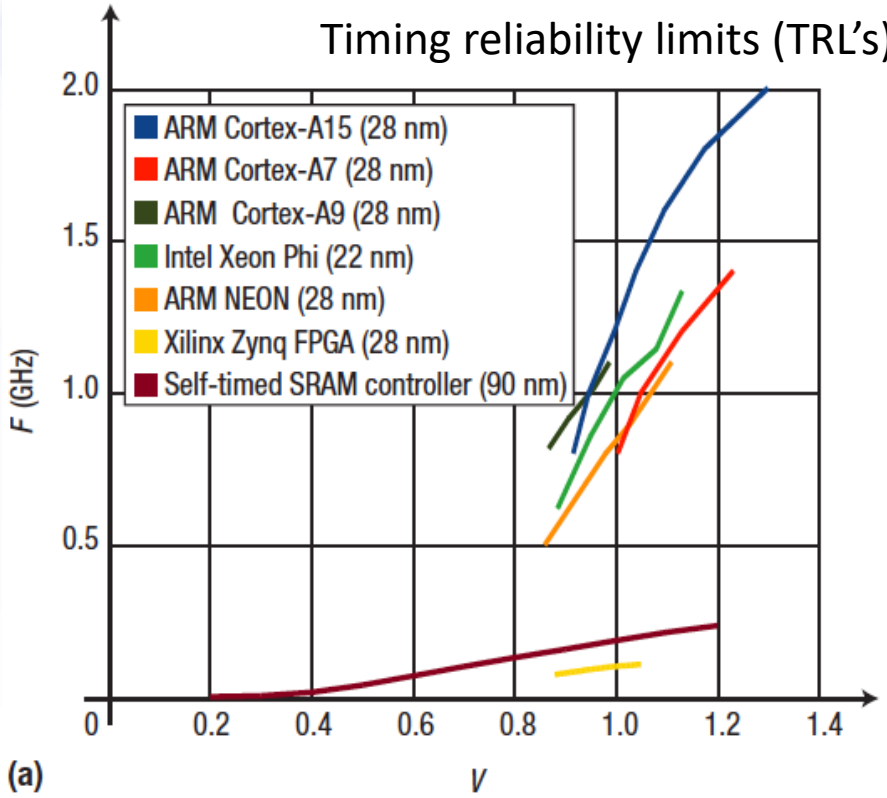


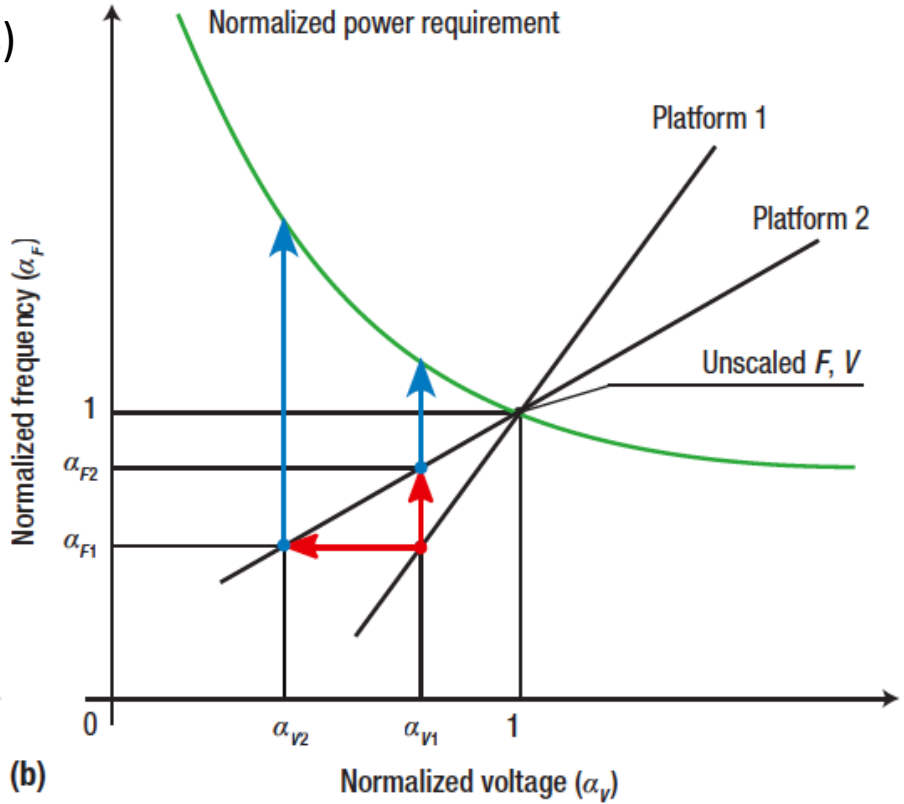
Effectiveness of parallelization

- Using parallelization to improve energy efficiency
- Compare energy efficiency *before* and *after* parallelization scaling
- Energy efficiency:
  - Power-normalized performance (IPS/Watt) - PNP
  - Operations per energy (Instructions/Joule) – same as PNP
  - Metric does not really care about performance
- H/W characteristics that affect this *before* vs *after* parallelization scaling 'effectiveness' η?

$$\eta = \frac{\theta_k / P_k}{\theta / P}$$

TRL's and binormalization





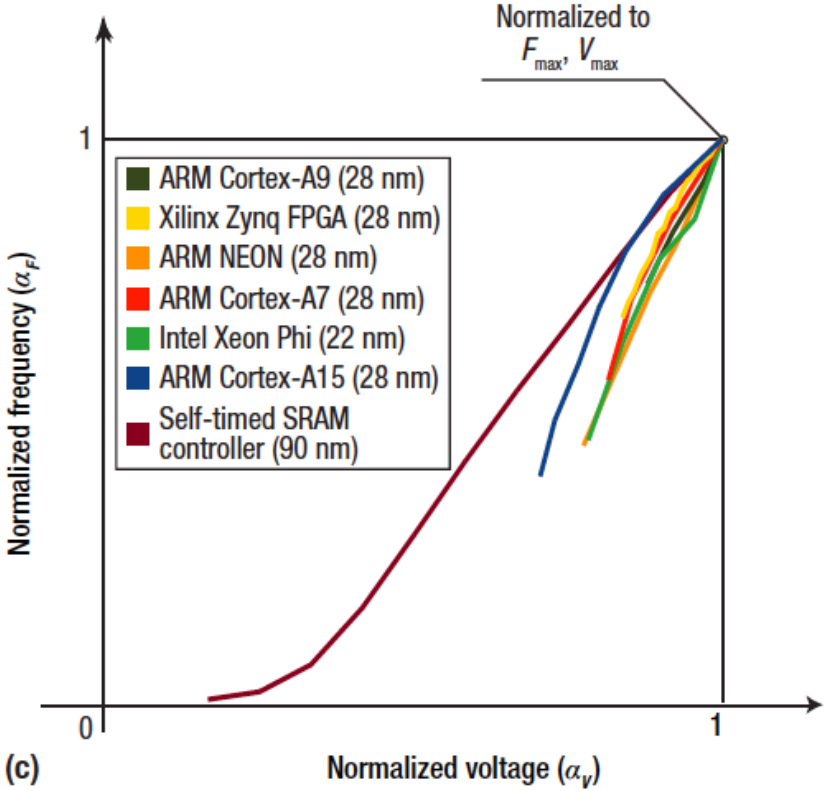
Effectiveness of parallelization

- The shallower a binormalized TRL, the more effective parallelization scaling would be for PNP

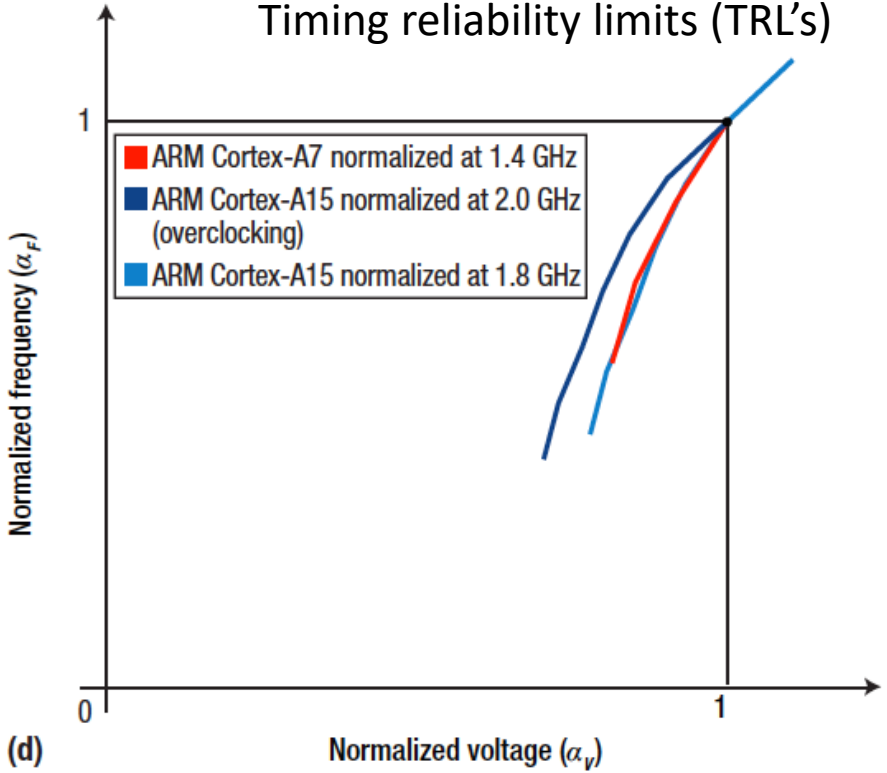
$$\eta = \frac{\theta_k / P_k}{\theta / P} = \frac{\left(k\alpha_F uF\right) / \left(kA\alpha_F F\alpha_V^2 V^2\right)}{\left(uF\right) / \left(AFV^2\right)} = \frac{1}{\alpha_V^2}$$

- What about EDP (energy-delay product)?
  - Basically re-emphasizing performance by multiplying throughput to PNP: $\vartheta^2/P = k\alpha_F/\alpha_V{}^2$ – the shallower the better

Binormalized TRL comparisons



Timing reliability limits (TRL's)



Complication 1:

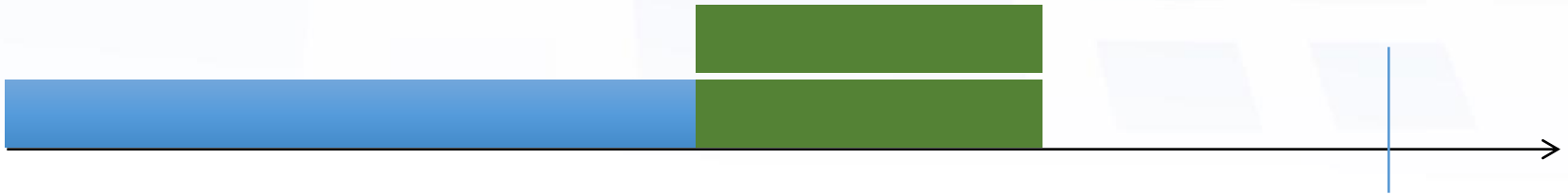
- How does your computation (algorithms) scale? Ideal scaling?
- Amdahl's Law and related models

Amdahl's Law

- Fixed workload
  - (50% parallelizable p=0.5)
  - On a sequential processor (single core) takes 1 unit of time to complete

Amdahl's Law

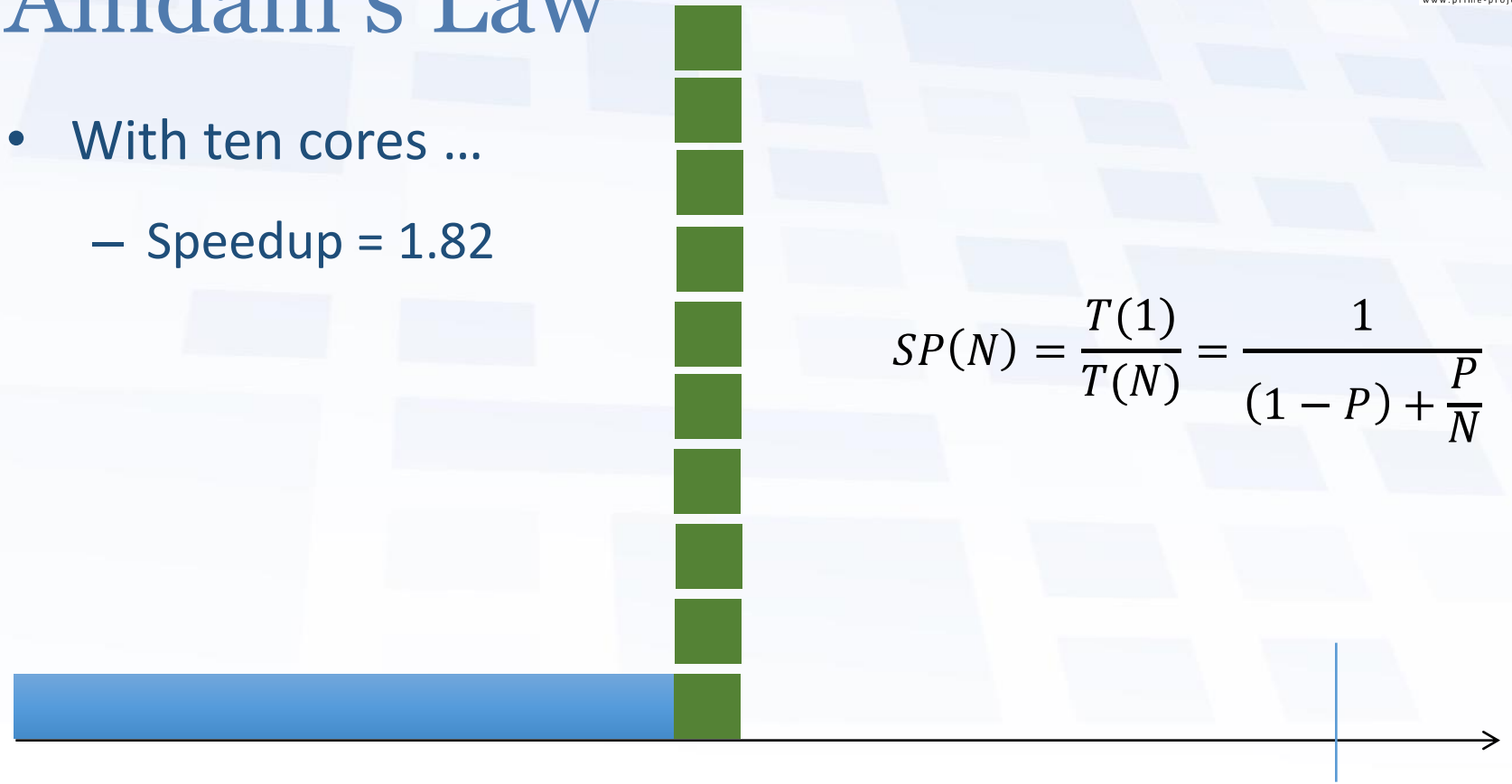
- With two cores …
  - Parallelizable part is distributed between the two cores
  - Total time 0.75
  - Speedup = 1/0.75 = 1.333



Amdahl's Law

- With ten cores …
  - Speedup = 1.82

$$SP(N) = \frac{T(1)}{T(N)} = \frac{1}{(1-P) + \frac{P}{N}}$$



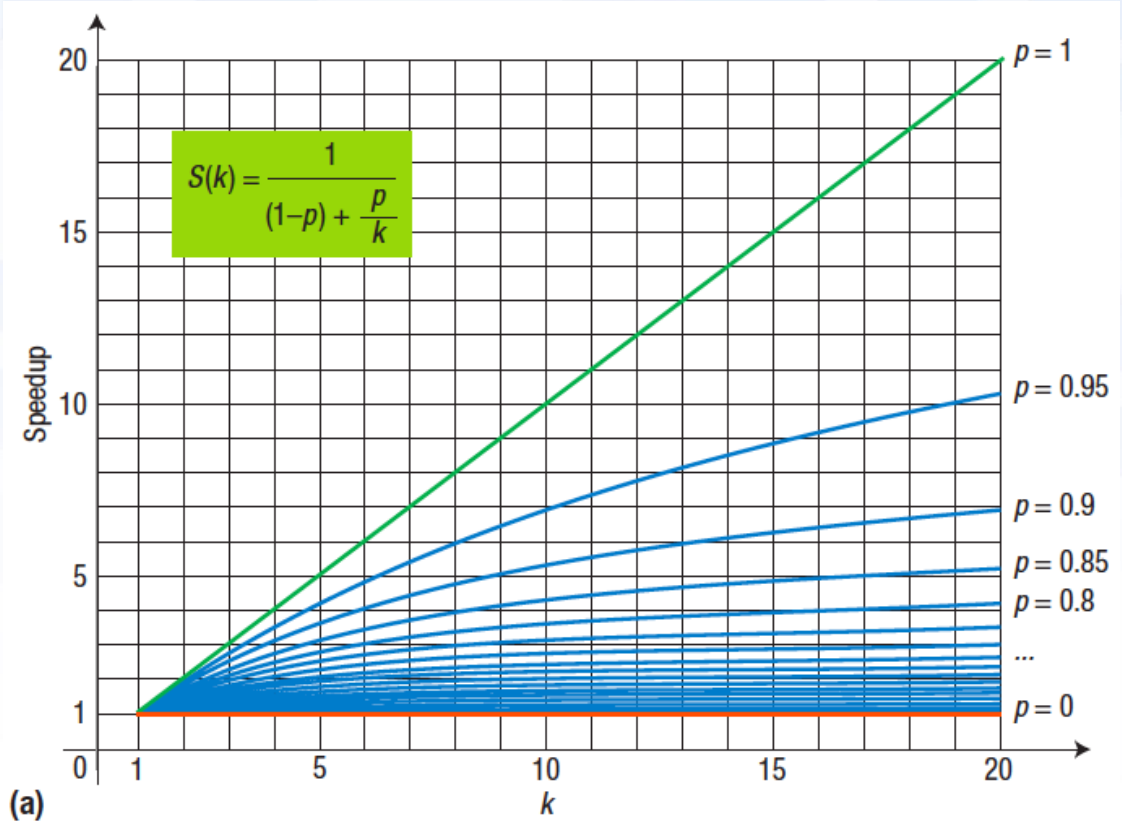
Amdahl's Law

- With ∞ cores …
  - Speedup = 2

$$SP(N) = \frac{T(1)}{T(N)} = \frac{1}{(1-P) + \frac{P}{N}}$$

$$SP(\infty) = \frac{1}{(1-P)}$$

Amdahl's Law


$S(k) = \frac{1}{(1-p) + \frac{p}{k}}$
Speedup
20
15
10
5
1
0
1
5
10
15
20
k
p = 1
p = 0.95
p = 0.9
p = 0.85
p = 0.8
...
p = 0
(a)

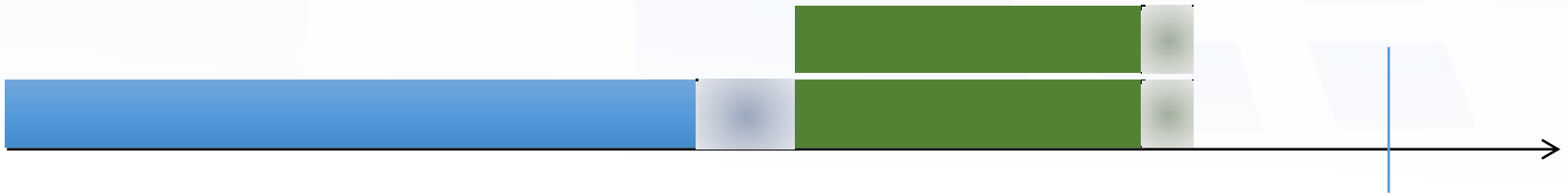
However

- These models deal with the parallelization and parallelizability of s/w on some hardware platform
  - *P* is a characteristic of some software
  - *N* is the number of h/w computation units that can be run in parallel
- Perfect mapping between s/w and h/w
  - No overhead or modification on speedup when mapping s/w to h/w
- In reality such overheads exist
  - System s/w and other s/w not belonging to the app under study
  - A platform's *N* units share synchronization because of shared memory, comms, etc.

M. Al-Hayanni et al., "Speedup and Parallelization Models for Energy-Efficient Many-Core Systems Using Performance Counters ," HPCS2017, Genoa, Italy, July 2017.

Modelling Basics

- Taking overheads into account, speedup can no longer be calculated from just knowing the application's *P* and the h/w's *N*
- Models must be able to contain the effects of overheads
- Here we deal with overheads in general, and system s/w overheads in particular



New speedup model

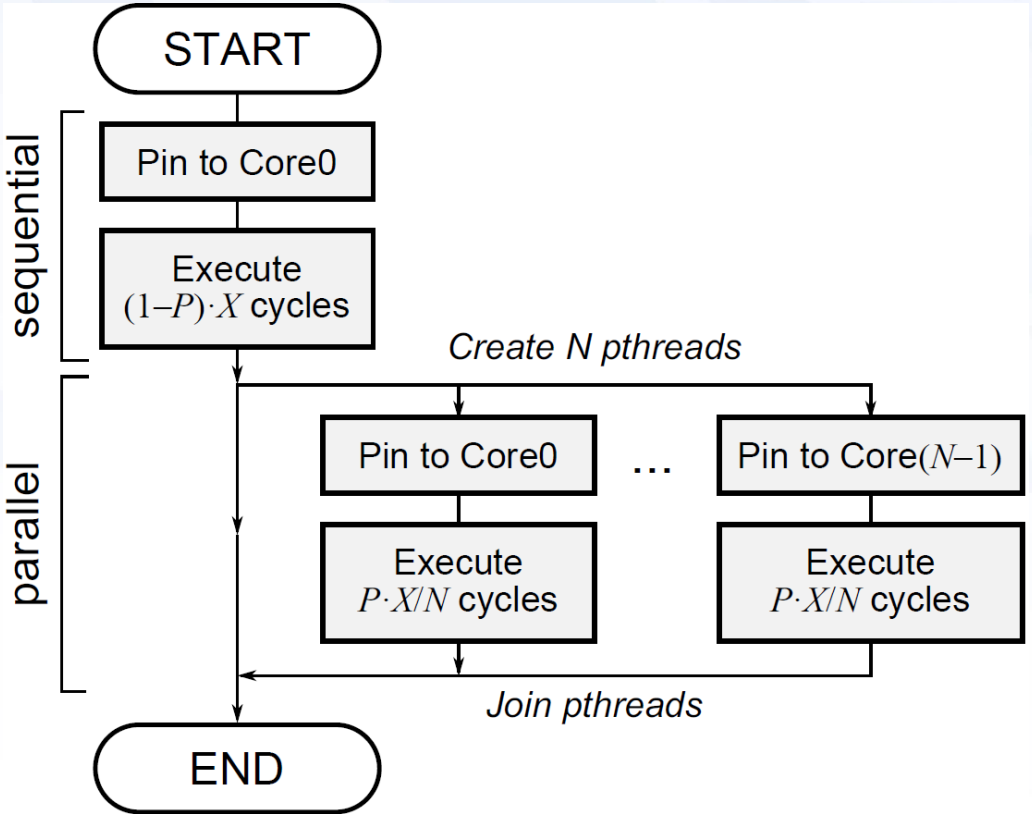
- An application's *P* is not as readily available as a platform's *N*
  - This seems to render Amdahl's Law less useful in the forward direction
- But using Amdahl's and similar models in the forward direction can be advantageous
  - For task-to-core matching, among other things, at runtime
- Being able to find an application's *P* is therefore a good thing
  - Need to consider overheads in such models because this detection of *P* is done with experiments
- Our models are based on regarding the total number of instructions as instructions belonging to the app under study I_0 and extra instructions ΔI from system s/w, memory and comms synchronizations, etc.

Determination of *P*

- If speedup can be measured
  - *P* can then be calculated if *N* is known
  - Execute an app on a platform with a known *N*
  - Measure the speedup
  - Find the *P* of executing this app on this platform
- But measuring speedup from execution time requires running an app at least twice
  - Not efficient at runtime
- Our models can be used to determine *P* from running the app once
  - Through recording certain performance counters

Experimental particulars

- Synthetic application
  - We developed a synthetic application which allows the s/w *P* value to be precisely set
  - Each 'cycle' includes some processor stress task (e.g. square root calculation) that requires no or min memory or comms
  - http://async.org.uk/data/speed-up-2016/



Experimental particulars

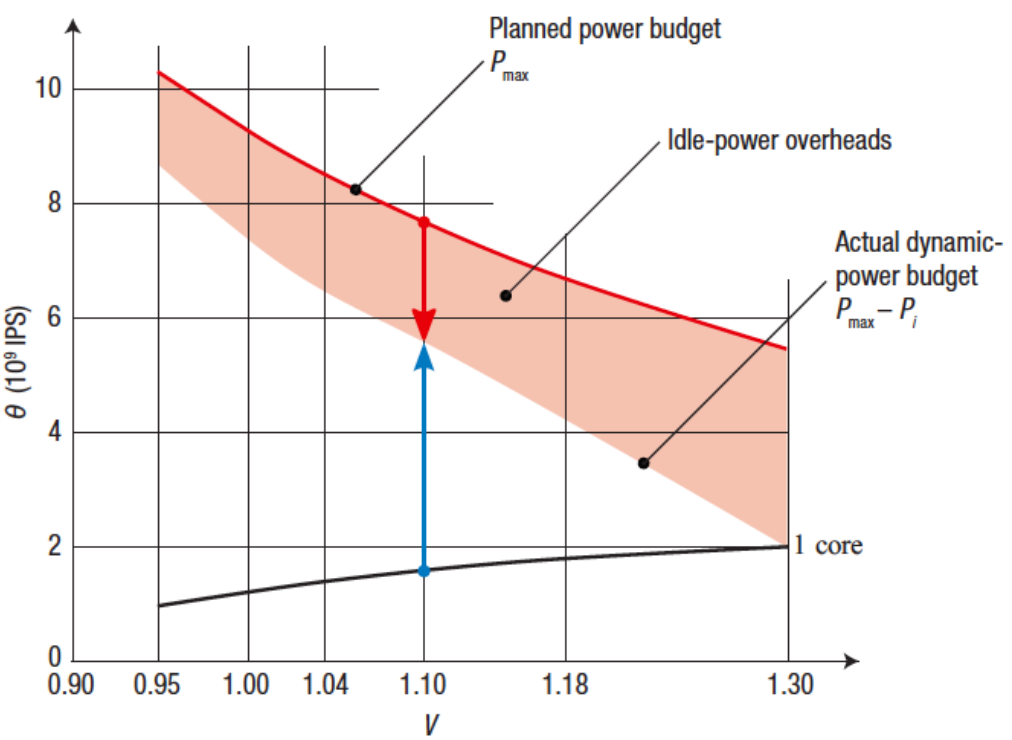
- Parsec applications
  - A total of nine Parsec applications are also used in this work
  - These cover a wide range of memory and CPU load characteristics
  - Cannot control *P* with Parsec apps
  - Experiments confirm that ΔI is more or less constant, regardless of application
  - In other words, ΔI is an overhead characteristic and not and an application characteristic – confirms the hypothesis

P-aware energy efficiency

- System energy efficiency is shown to be related to *P* and *N*
  - Opening up opportunities for *P*-aware task-to-core mapping and other runtime controls
  - We investigated beyond the intuitive ‘if an app has a low *P* there is no point giving it a lot of cores’
  - Two metrics investigated
    - PNP (energy efficiency) and EDP (energy efficiency + throughput)

Complication 2:

- It's not just switching power alone
- Not only throughput has overheads – power too
- Best with experimental data rather than models
- Tool at http://async.org.uk/prime/PER/
- Video featured on IEEE Computer Society webpage



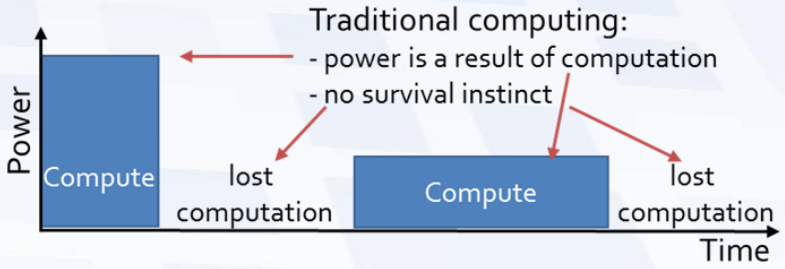
Power/energy-bound

- Existing approach:
  - Real-time design paradigm (throughput-bound systems)
  - Power taken care of by ‘as cool as possible’ via low-power techniques
  - Timing correctness is the big aim
- New approach:
  - Real-power design paradigm (power/energy-bound systems)
  - Throughput taken care of by ‘as fast as possible’ via high efficiency techniques
  - Amount of computation taken care of by ‘as much as possible’ via high efficiency techniques
  - Quality can be traded for energy
  - Autonomy and survivability are the big aims

Power/energy-bound

- **Typical requirements**
  - Performance

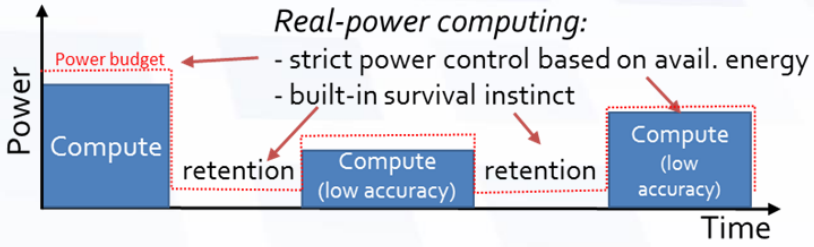
Minimise power/energy,
while meeting performance



Traditional low-power computing systems do not automatically provide autonomy

- **Typical requirements**
  - Power/Energy/Quality
  - Performance

Elastically control computation quality
based on energy and performance



Power-constrained computing ensures autonomy and energy-efficiency

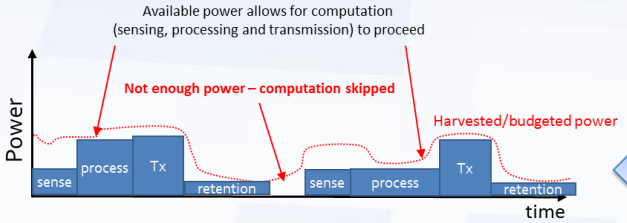
Key challenges

- Full energy transparency at all levels of abstraction
  - Electronics, compute and communicate units, programming constructs
- Higher design productivity and robustness
  - For systems with power/energy constraints
- Power and energy monitoring and control at runtime
  - Uncertainties for energy/power orders of magnitude higher than for throughput
- Holistic h/w design for energy/quality tradeoffs
  - New power-regulation and computational approaches needed
- Migrating from energy efficiency to energy effectiveness
  - Ability to convert energy into computation in a wide band of power-supply conditions

Envisioned methodology

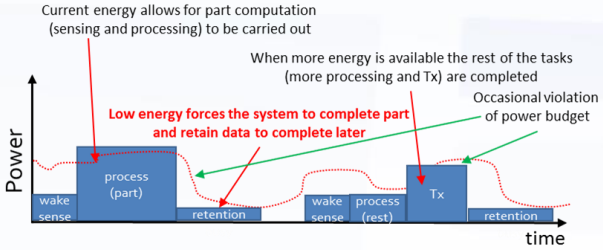
- Power-centric control of resources
  - Interacting external interfaces, computation (hardware/software) tasks and underlying subsystems
- Power-compute codesign
  - Design-time co-optimization, power-delivery policies for survivability
- Runtime adaptation for survivability
  - Continuous optimization for energy efficiency using control levers and monitors
- Interplay between design-time activities and runtime support
  - For optimal resource allocation policies
  - One cannot succeed without the other

Real-power paradigm



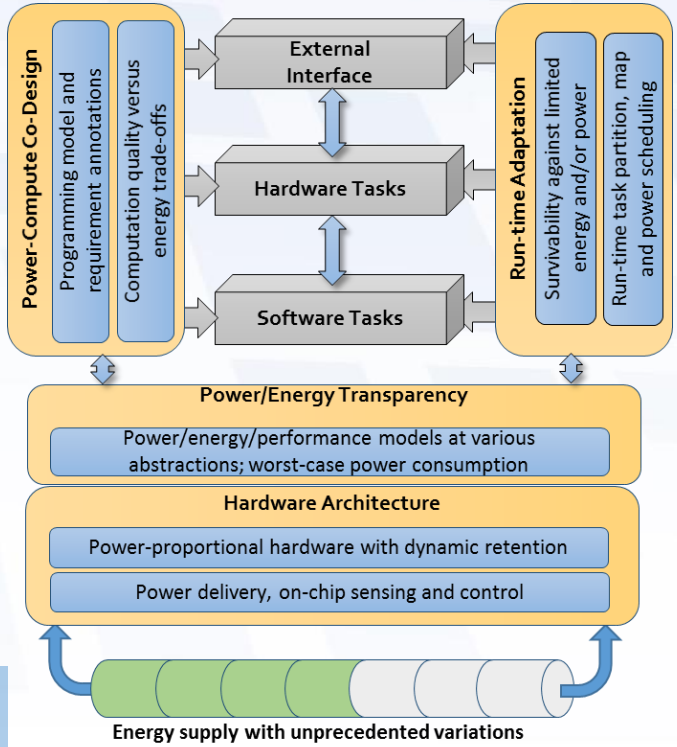
Hard real-power computing

- No battery/no storage
- Extensive power-compute co-design needed



Soft real-power computing

- With energy storage
- Power-compute co-design
- + run-time adaptation



R. Shafik and A. Yakovlev, Real-Power Computing talk at the ARM summit, Cambridge 11th Sep 2017

Thank you

PRiME
www.prime-project.org