# Two short talks on current topics in Computer Science

Jan Prins

Department of Computer Science
University of North Carolina at Chapel Hill

1. Runtime Methods to Improve Energy Efficiency in Supercomputing Applications
2. Computational Methods in Transcriptome Analysis

# Runtime Methods to Improve Energy Efficiency in HPC Applications

Sridutt Bhalachandra[1], Robert Fowler[2], Stephen Olivier[3], Allan Porterfield[2], and Jan Prins[1]

[1]Department of Computer Science, University of North Carolina at Chapel Hill
[2]Renaissance Computing Institute, Chapel Hill
[3]Sandia National Laboratories

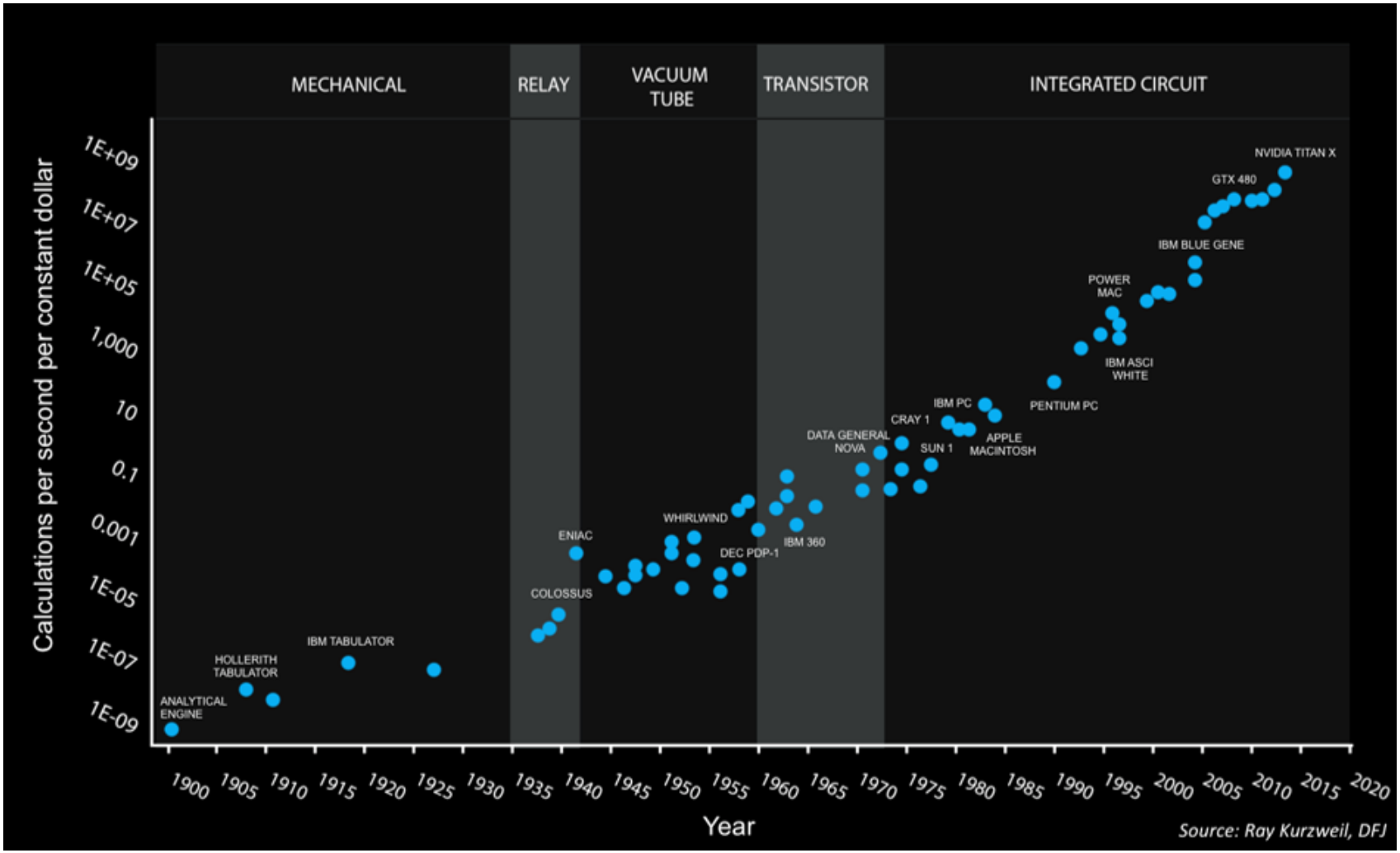
May 8, 2018

THE UNIVERSITY of NORTH CAROLINA at CHAPEL HILL

renci

Sandia National Laboratories

# computing performance: 120 years of exponential growth!



# What is driving performance growth?

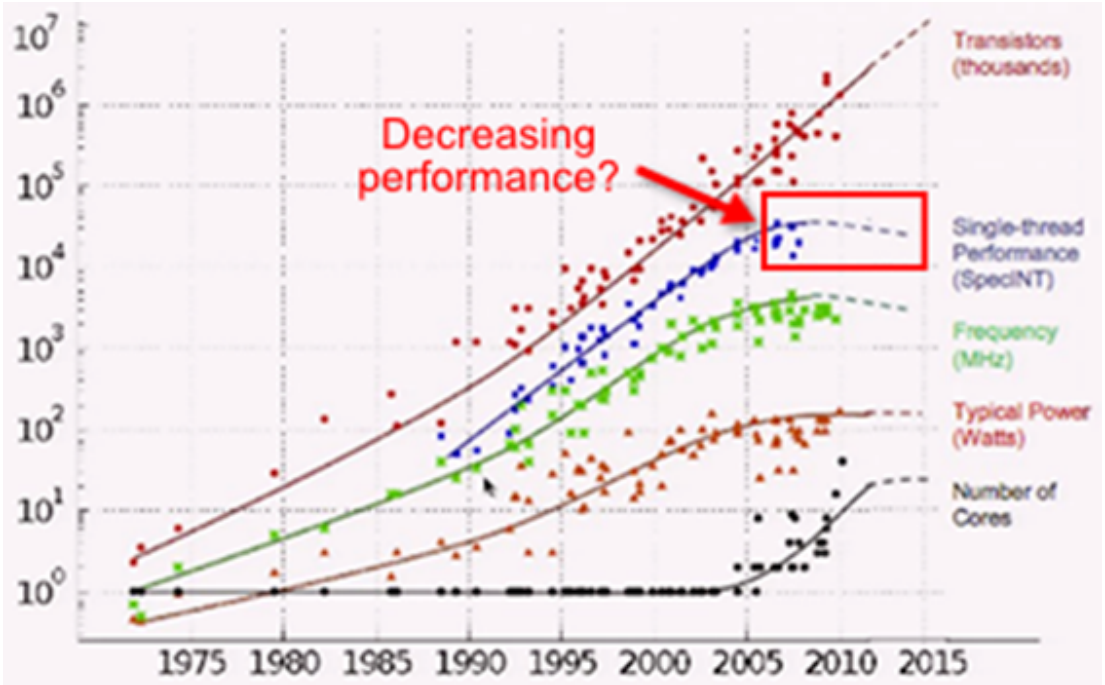
- *Moore's "law"* - transistor density doubles every 18-24 months

# What is driving performance growth?

- *Moore's "law"* - transistor density doubles every 18-24 months
- *Dennard scaling* - total power remains the same and maximum operating frequency increases

# What is driving performance growth?

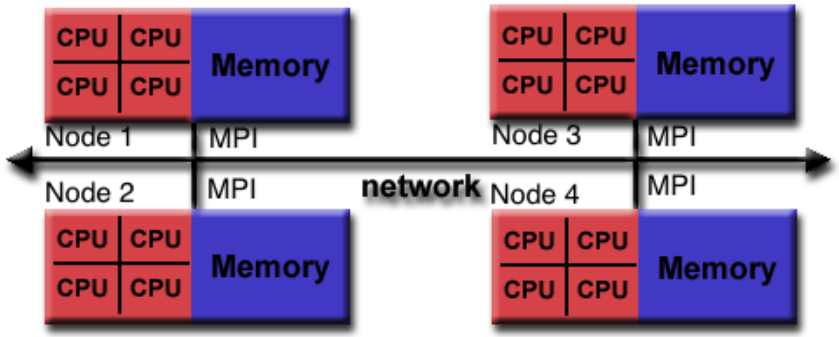
- *Moore's "law"* - transistor density doubles every 18-24 months
- *Dennard scaling* - total power remains the same and maximum operating frequency increases
- but look what has happened over the past two decades



# The end of Dennard scaling and faster transistors

Consequences

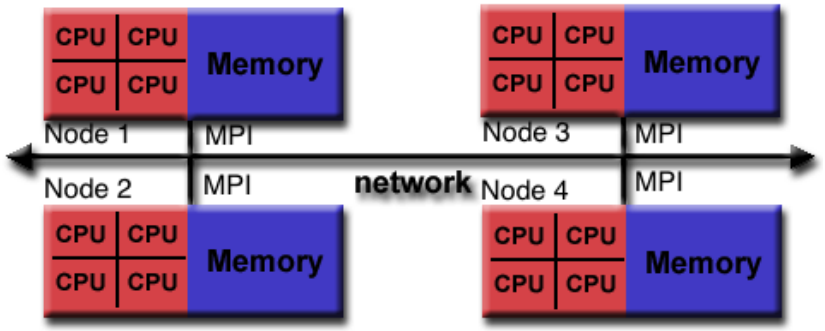
- additional transistors require additional area
- power and heat increase commensurately
- parallel computing is the only route to scaling performance

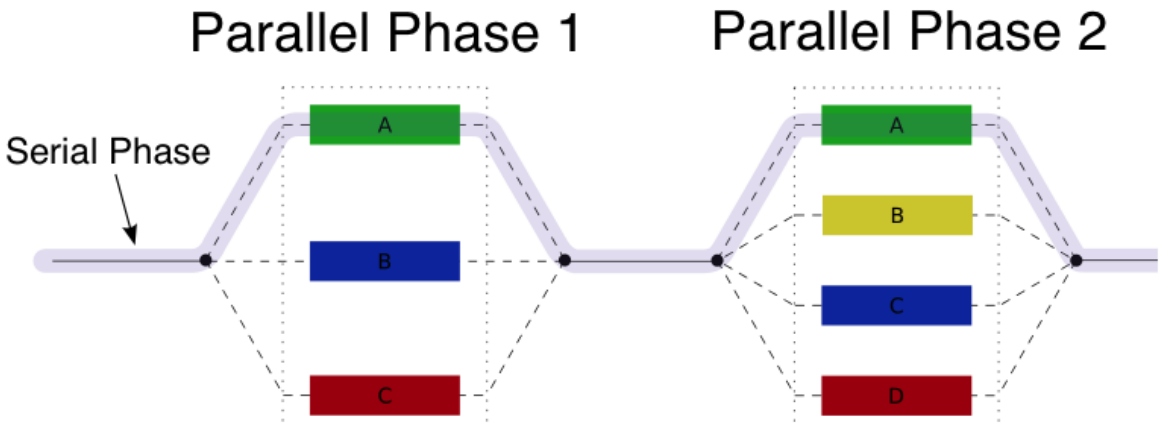



multicore processors
multiprocessor nodes
interconnection networks

# Scalable parallel computing

Message Passing Interface (MPI) is used to coordinate computation and communication among all processor cores



# Current largest parallel computer


Source: http://www.nsccwx.cn/wxcyw

## Sunway Taihulight

40,960 nodes
10,649,600 cores (256+4 per node) at 1.45GHz
20PB storage
$273 million
Top500 #1
**93.01 PFLOPS @ 15.4MW**

- 1 PetaFLOPS (PFLOPS) = $10^{15}$ Floating Point Operations Per Second
- 1 MegaWatt (MW) can roughly power 1000 homes

# Exascale ($10^{18}$ *FLOPS*) power requirements

| System/Site | Performance (PFLOPS) | Power (MW) | Energy Efficiency (GFLOPS/W) |
|---|---|---|---|
| Exascale | 1000 | ? | ? |
| Taihulight | 93 | 15 | 6 |
| Tianhe 2 | 34 | 18 | 2 |
| Piz Daint | 20 | 2 | 9 |

# Exascale ($10^{18}$ *FLOPS*) power requirements

| System/Site | Performance (PFLOPS) | Power (MW) | Energy Efficiency (GFLOPS/W) |
|---|---|---|---|
| Exascale | 1000 | ? | ? |
| Taihulight | 93 | 15 | 6 |
| Tianhe 2 | 34 | 18 | 2 |
| Piz Daint | 20 | 2 | 9 |
| TSUBAME 3.0 | 2 | 0.14 | 14 |
| kukai | 0.46 | 0.03 | 14 |
| AIST AI Cloud | 0.96 | 0.08 | 13 |

# Exascale ($10^{18}$ FLOPS) power requirements

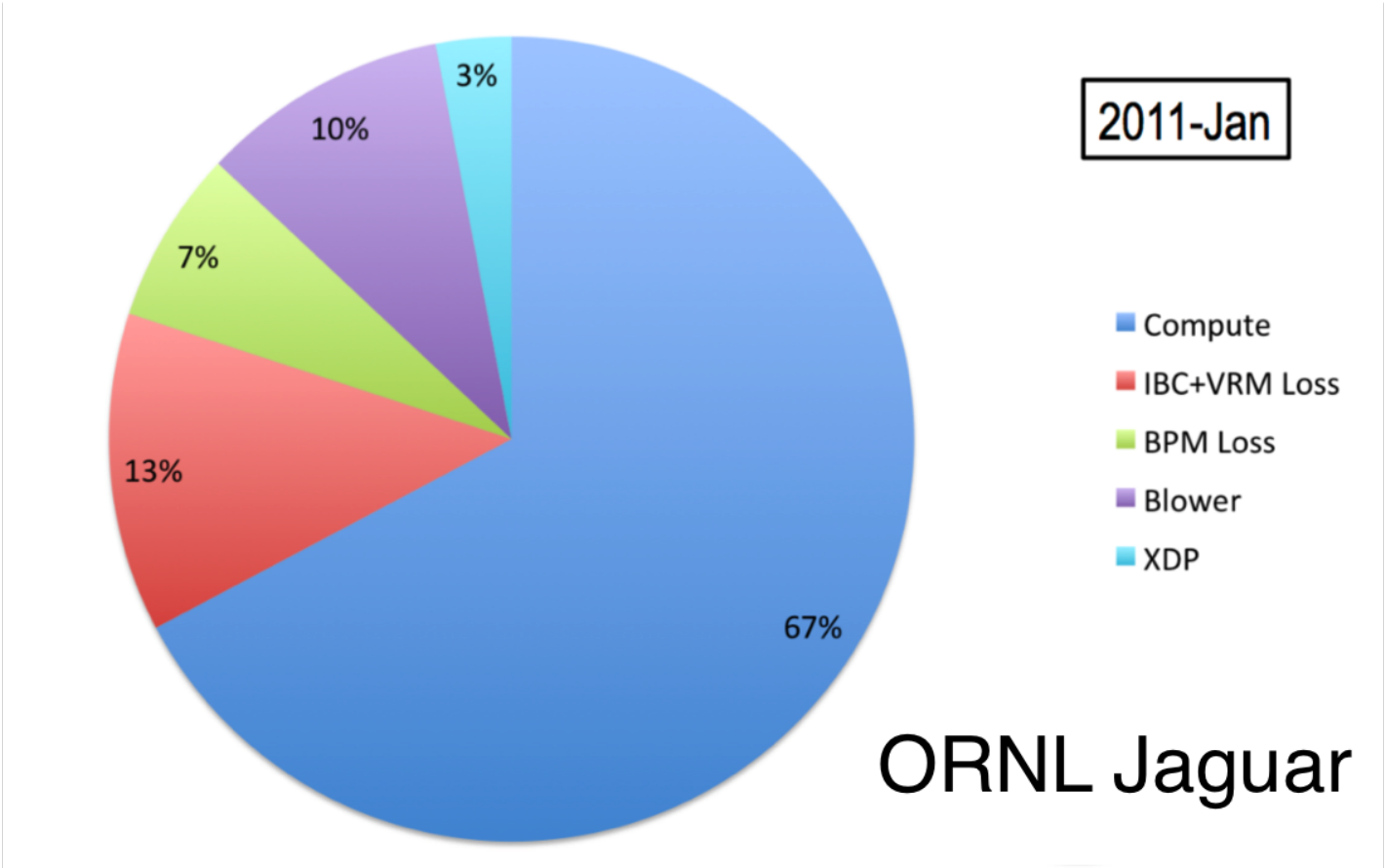
| System/Site | Performance (PFLOPS) | Power (MW) | Energy Efficiency (GFLOPS/W) |
|---|---|---|---|
| Exascale | 1000 | 20 | 50 |
| Taihulight | 93 | 15 | 6 |
| Tianhe 2 | 34 | 18 | 2 |
| Piz Daint | 20 | 2 | 9 |
| TSUBAME 3.0 | 2 | 0.14 | 14 |
| kukai | 0.46 | 0.03 | 14 |
| AIST AI Cloud | 0.96 | 0.08 | 13 |

# Exascale ($10^{18}$ FLOPS) power requirements

| System/Site | Performance (PFLOPS) | Power (MW) | Energy Efficiency (GFLOPS/W) |
|---|---|---|---|
| Exascale | 1000 | 20 | 50 |
| Taihulight | 93 | 15 | 6 |
| Tianhe 2 | 34 | 18 | 2 |
| Piz Daint | 20 | 2 | 9 |
| TSUBAME 3.0 | 2 | 0.14 | 14 |
| kukai | 0.46 | 0.03 | 14 |
| AIST AI Cloud | 0.96 | 0.08 | 13 |

**5x - 10x improvement in energy efficiency required**

# breakdown of power use in a large parallel computer

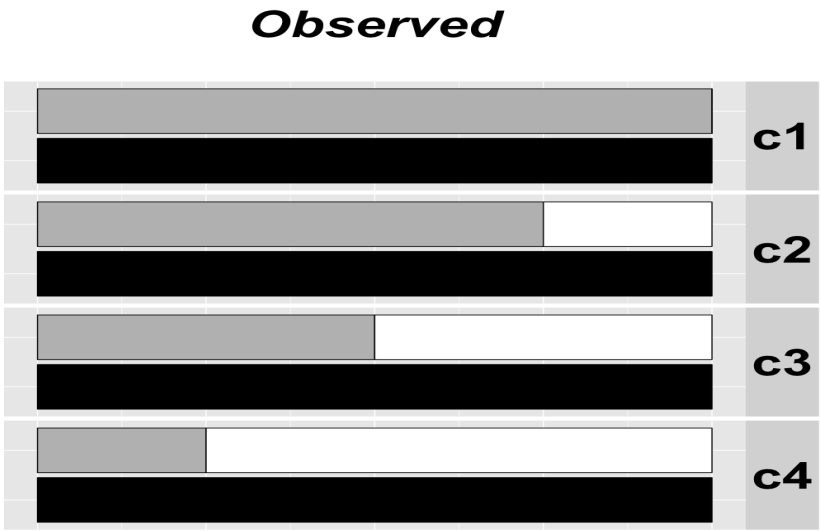



Source: Use Case: Quantifying the Energy Efficiency of a Computing System -Hsu et al.

# Opportunity to save energy

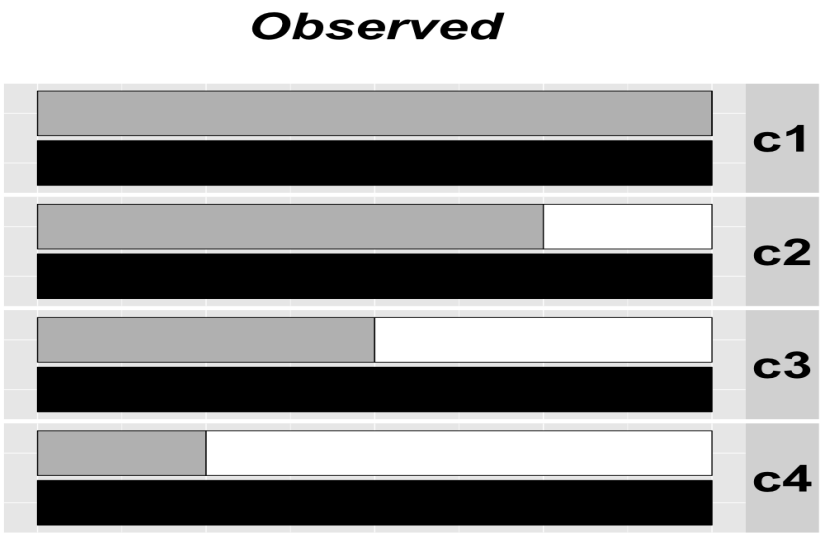
"Race to the end" in parallel regions

- each processor core operates on data in its node
- each processor maximizes speed while staying within thermal limit
- all processors spinwait on lock at end of the region
- last processor to arrive releases the lock

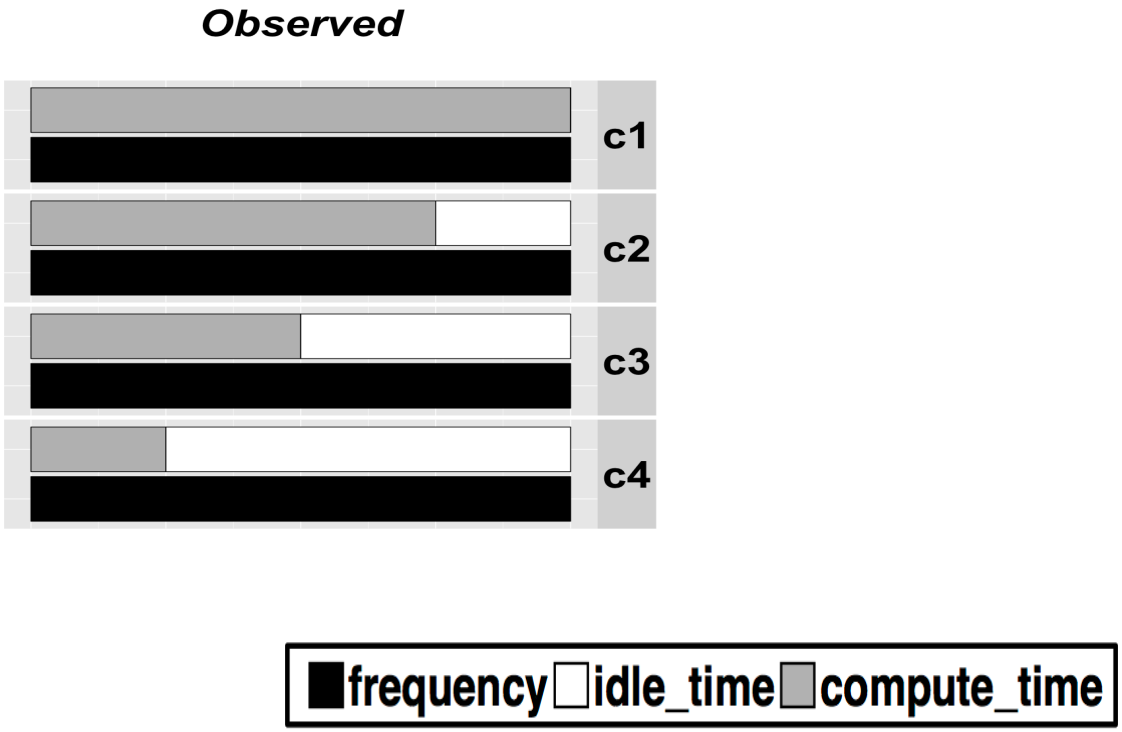


# Computational workload imbalance

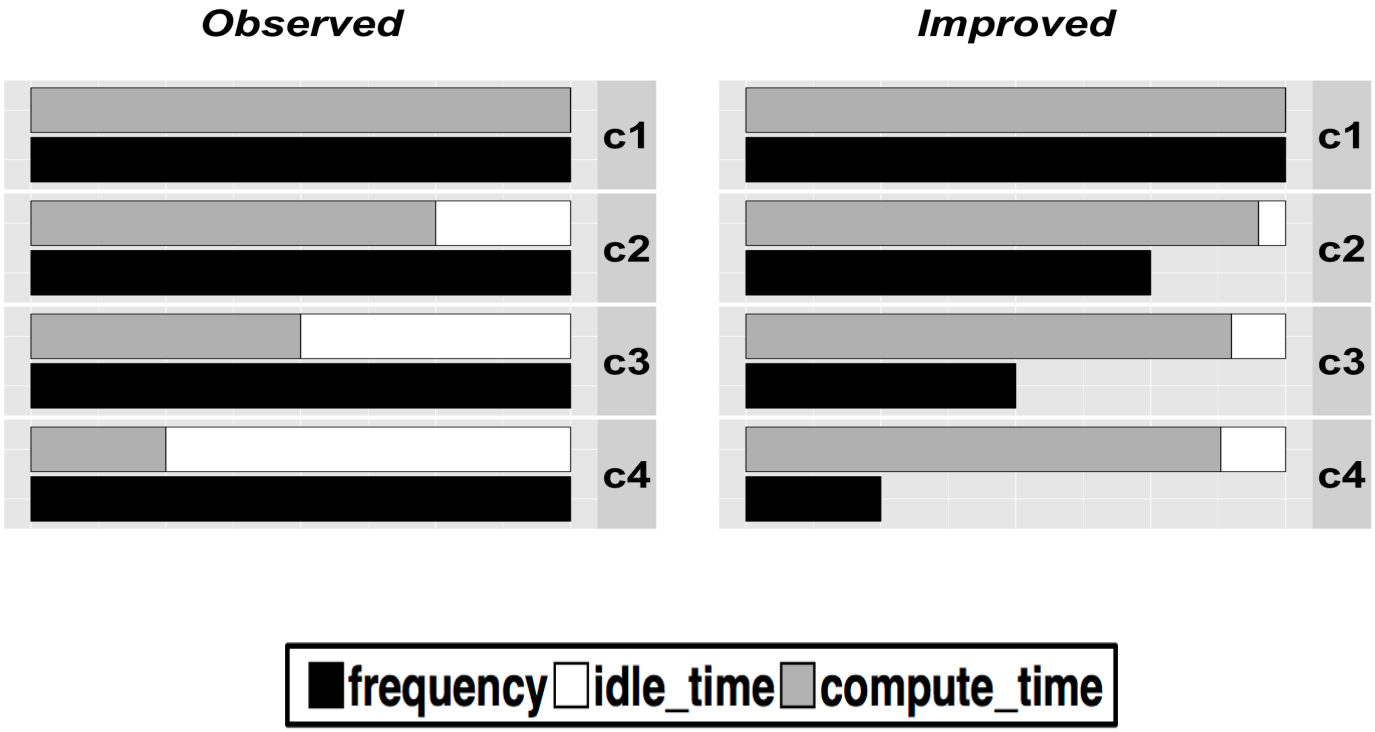
- could be inherent in application
- could be due to system heterogeneity
- exacerbated by the race to the end



# Saving energy by mitigating workload imbalance



# Saving energy by mitigating workload imbalance

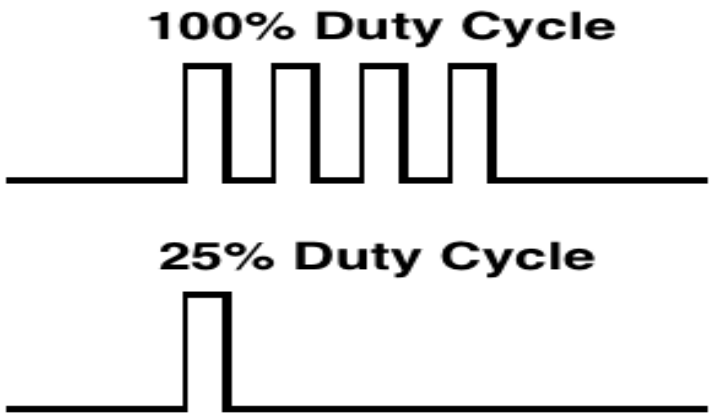



## Challenges

- each core is set to operate at a suitable frequency based on previous phase observation
- the frequency can change at every phase

# Fine grained power control

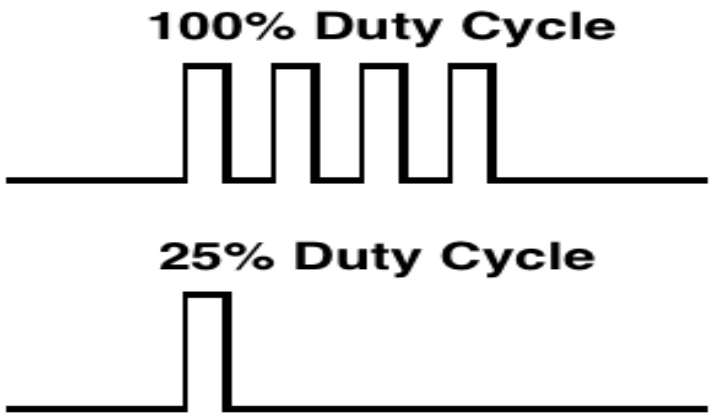
- Dynamic Duty Cycle Modulation (DDCM) – T-states




  - Actual clock rate is not changed, DVFS and TurboBoost still operational
  - Modulation range constant across architecture - 100% to 6.25%
  - **IA32_CLOCK_MODULATION** MSR

# Fine grained power control

- Dynamic Duty Cycle Modulation (DDCM) – T-states




  - Actual clock rate is not changed, DVFS and TurboBoost still operational
  - Modulation range constant across architecture - 100% to 6.25%
  - **IA32_CLOCK_MODULATION** MSR

- DVFS - core specific (Haswell) – P-states
  - Can slow only non-critical cores
  - Operational range machine-dependent even for the same architecture
  - **acpi_cpufreq** kernel module

# Runtime control policy

## Core-specific control

- match a core's effective duty cycle to its workload

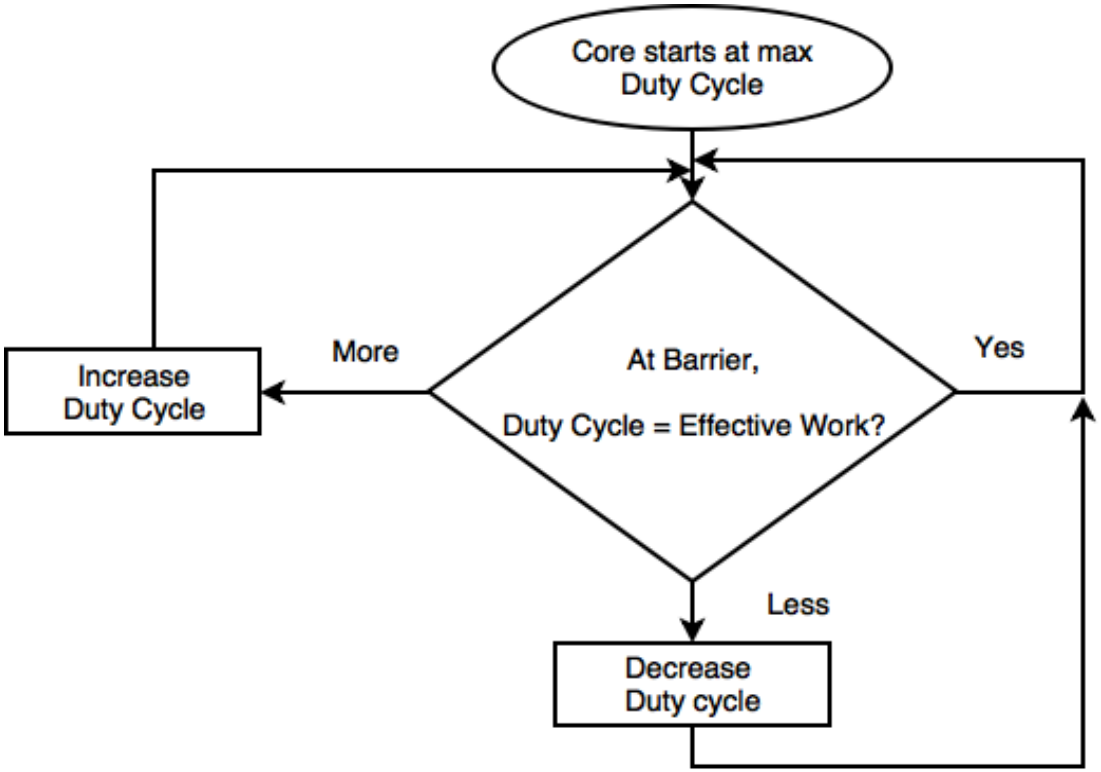
$$Duty\ cycle = \frac{Time\ core\ in\ active\ state}{Total\ time\ (clock\ cycles)}$$

∗Change core active time using DDCM or clock cycles using DVFS

$$Work = \frac{Compute\ time}{Compute\ time + Idle\ time}\ (constant\ frequency)$$

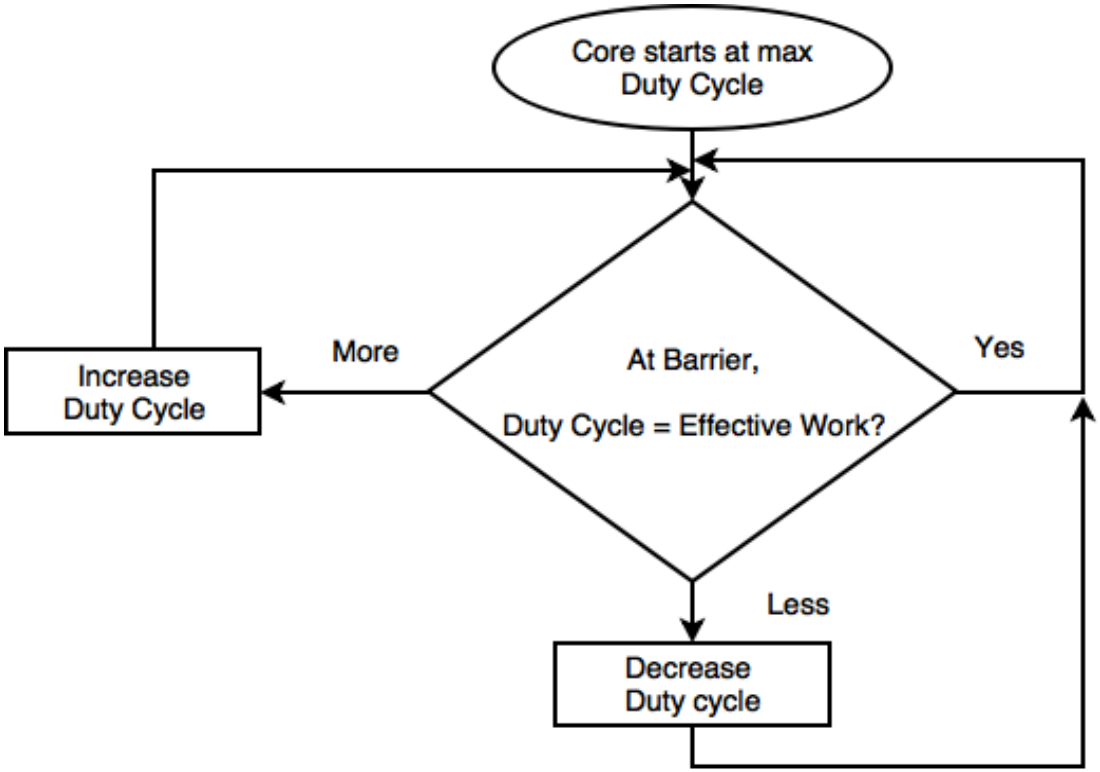
$$Effective\ Work = \frac{Compute\ time}{Compute\ time + Idle\ time} * \frac{Max\ frequency}{Current\ frequency}$$

# Runtime policy




- Assumes similar behavior across successive phases
- Policy calculation local to core, no communication

# Runtime policy




- Assumes similar behavior across successive phases
- Policy calculation local to core, no communication

**Combined policy**

($Power_{DVFS} < Power_{DDCM}$)

- Use DVFS policy until lowest frequency reached
- Thereafter, use DDCM policy



# ACR = Runtime Policy + User Options

1. Can monitor performance degradation at the end of every phase
   - Rudimentary method to detect phase change
2. Can induce minimum phase length limit
   - Useful in skipping start-up phases
3. Support for user-annotations
   - However, not used in current experimentation

* Runtime is transparent, eliminating the need for code changes to MPI applications



# Mini-apps & Applications

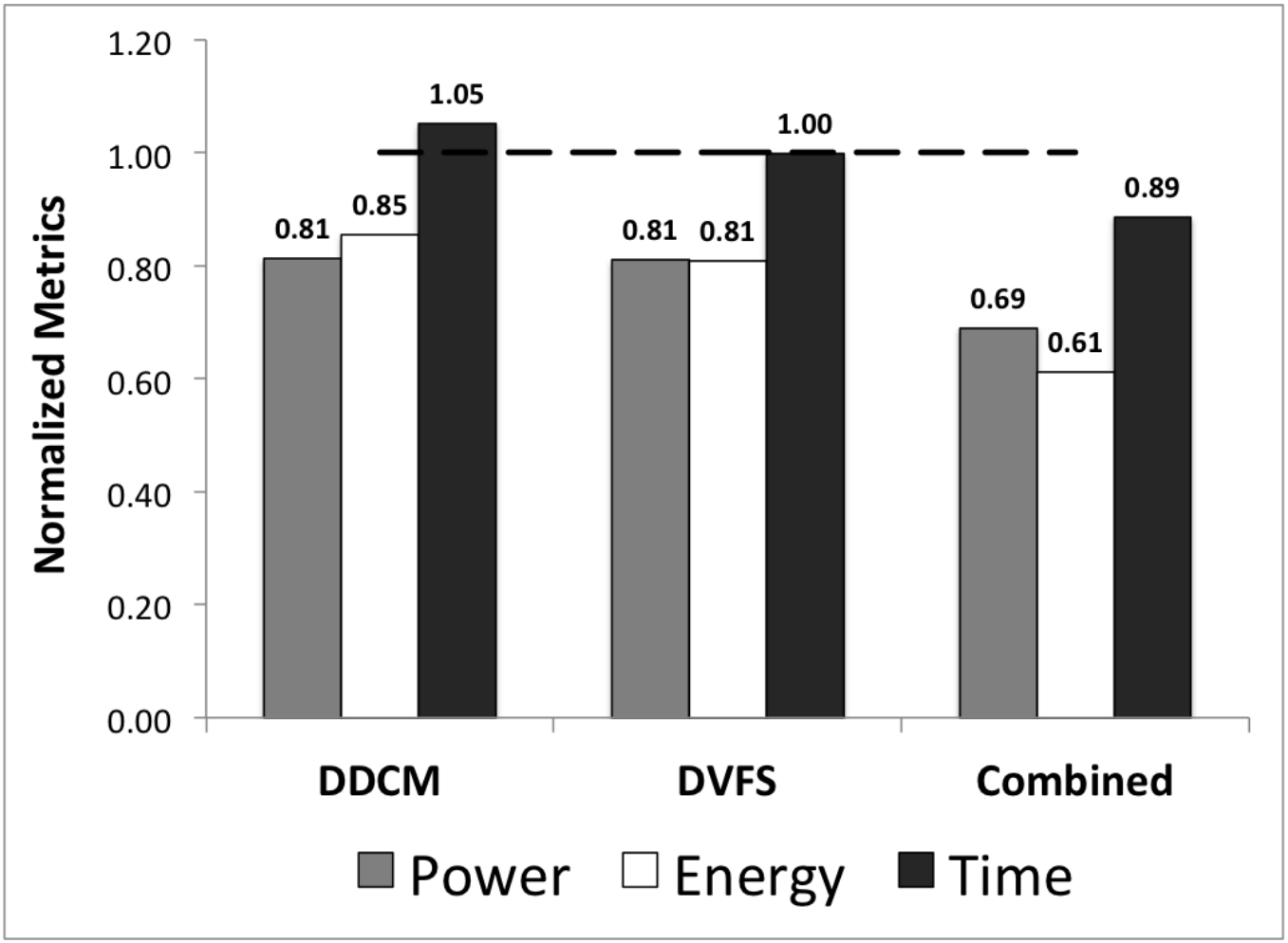
- Unstructured grids – MiniFE, HPCCG, AMG
- Structured grids – MiniGhost
- Mesh Refinement – MiniAMR
- Hydrodynamics – CloverLeaf
  - mini-apps representative of key production HPC applications
- **Dislocation Dynamics – ParaDis**

# System

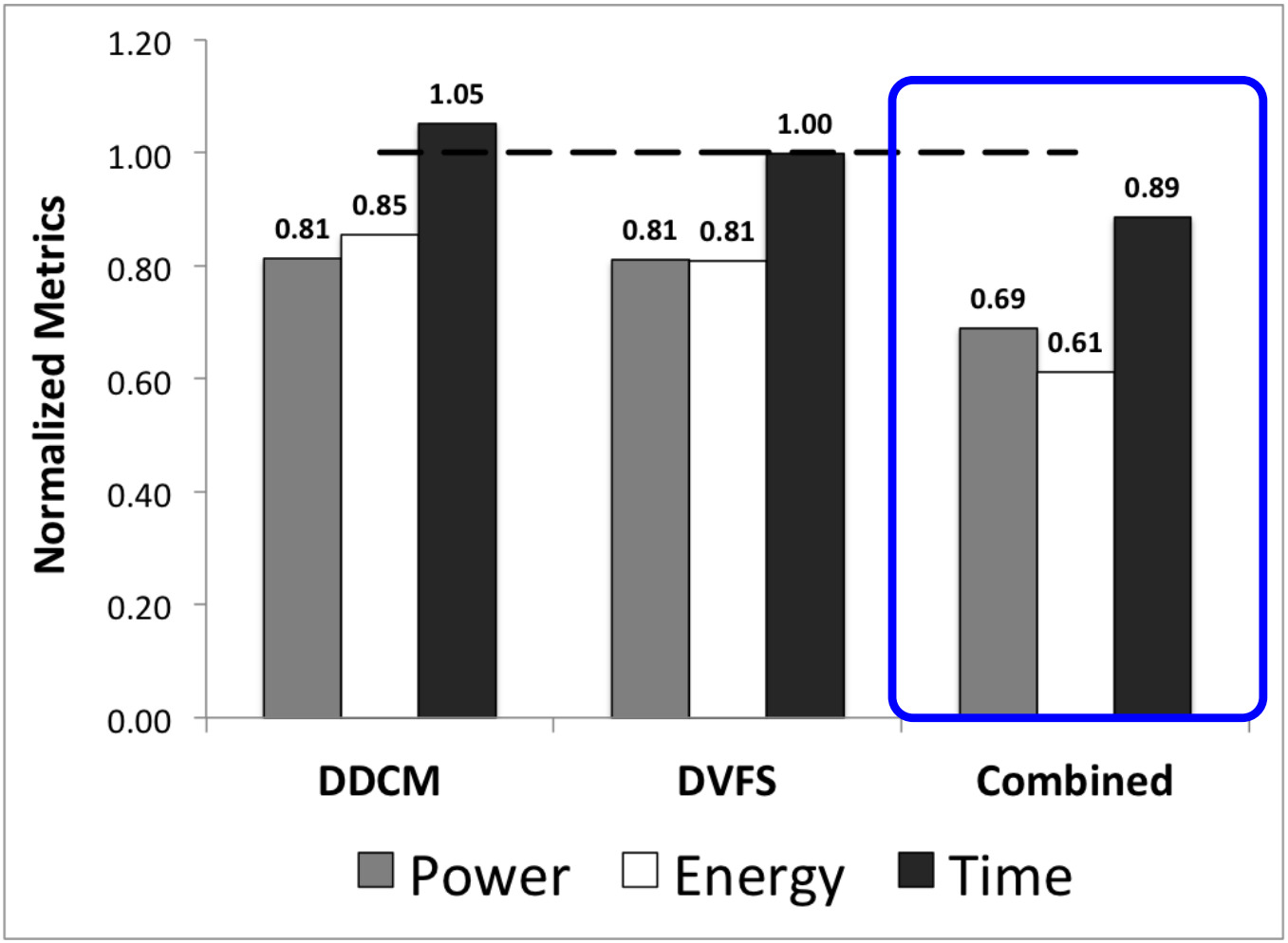
- 32 Haswell node partition (Sandia Shepard) = 1024 cores
  - Dell M420: two 16-cores Xeon E5-2698v3 128GB at 2.3GHz
  - RHEL6.8, Slurm 2.3.3-1.18chaos and Linux 3.17.8 kernel
  - Mpich 3.2

Results are average of 12 runs taken at stable temperatures (to promote reproducibility)

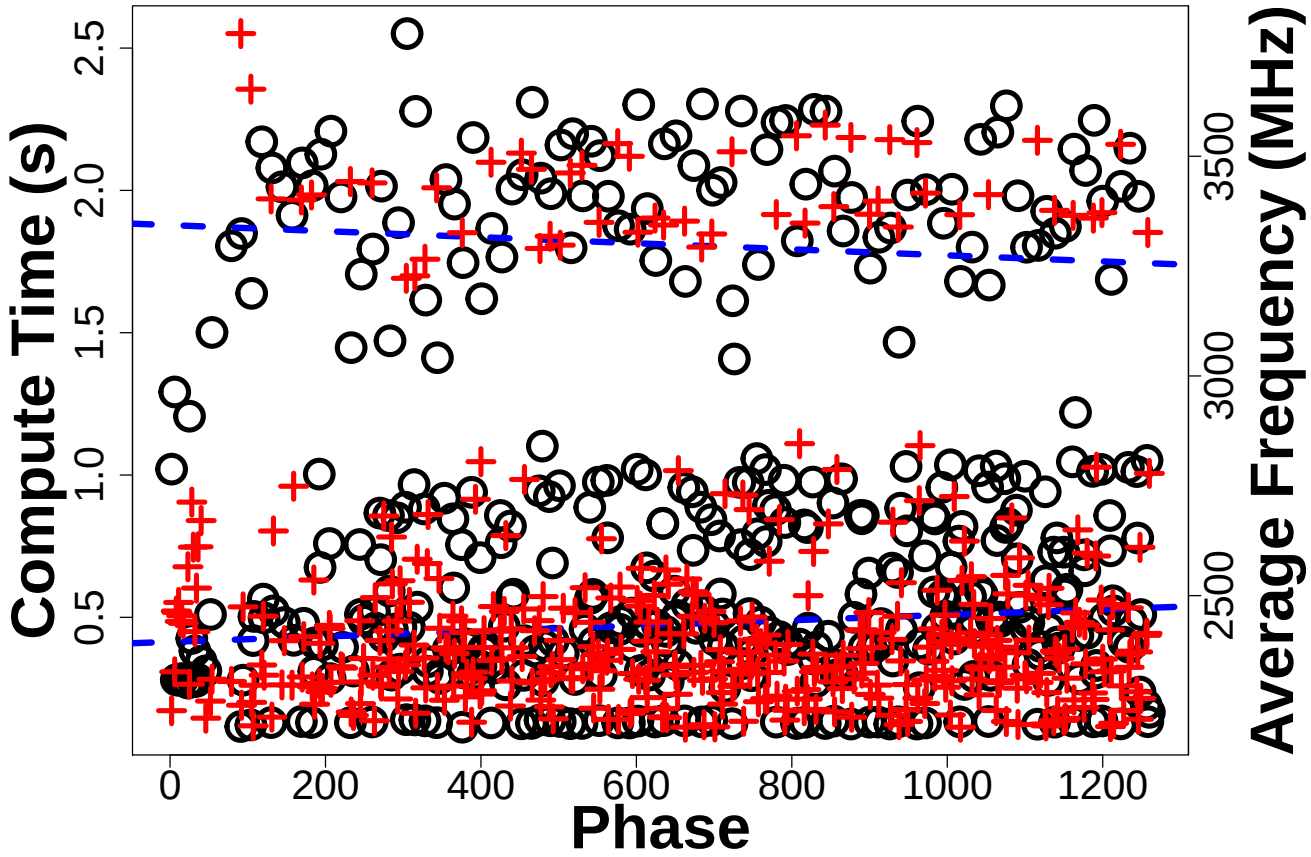
# *ParaDis* results



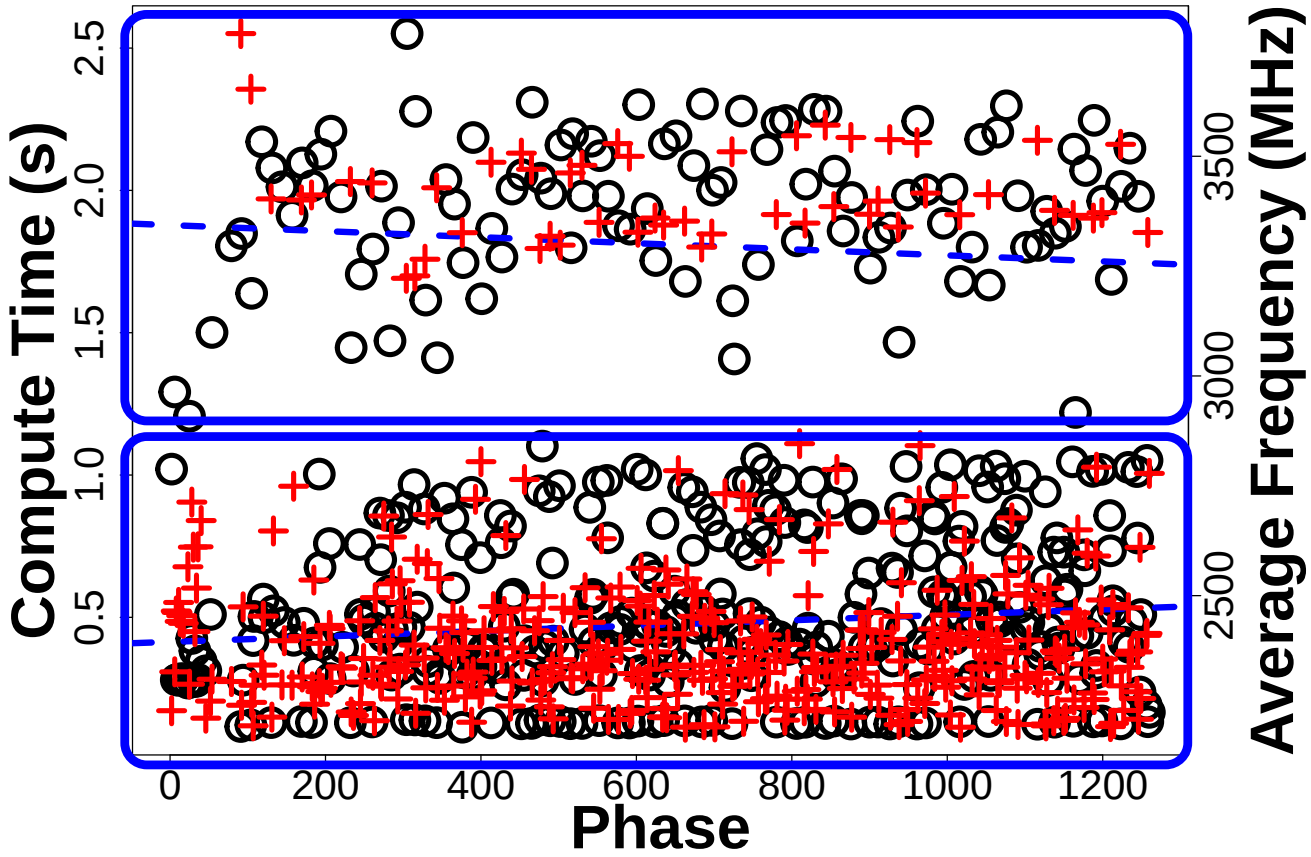
# *ParaDis* results



# *ParaDis* critical path on 24 nodes (768 cores) - Default

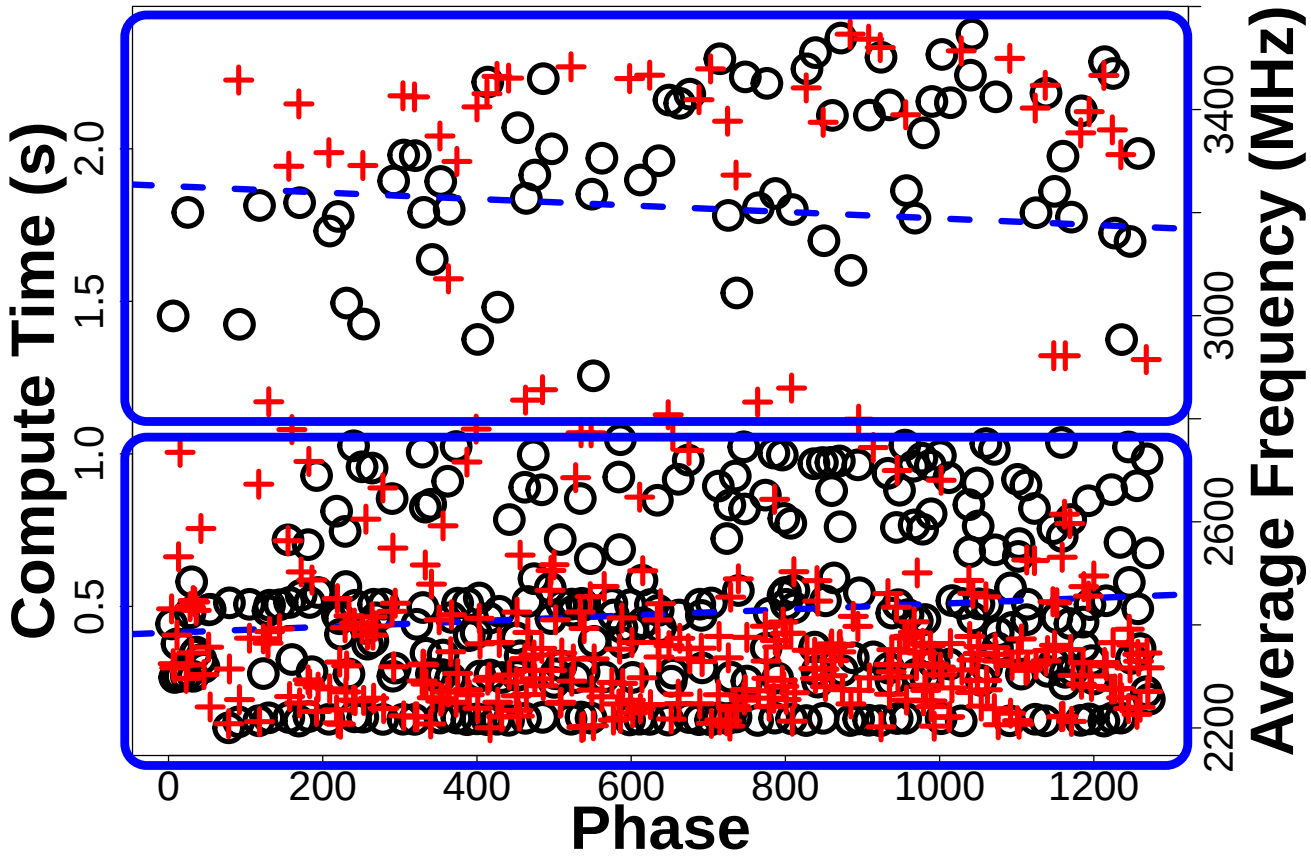


# *ParaDis* critical path on 24 nodes (768 cores) - Default



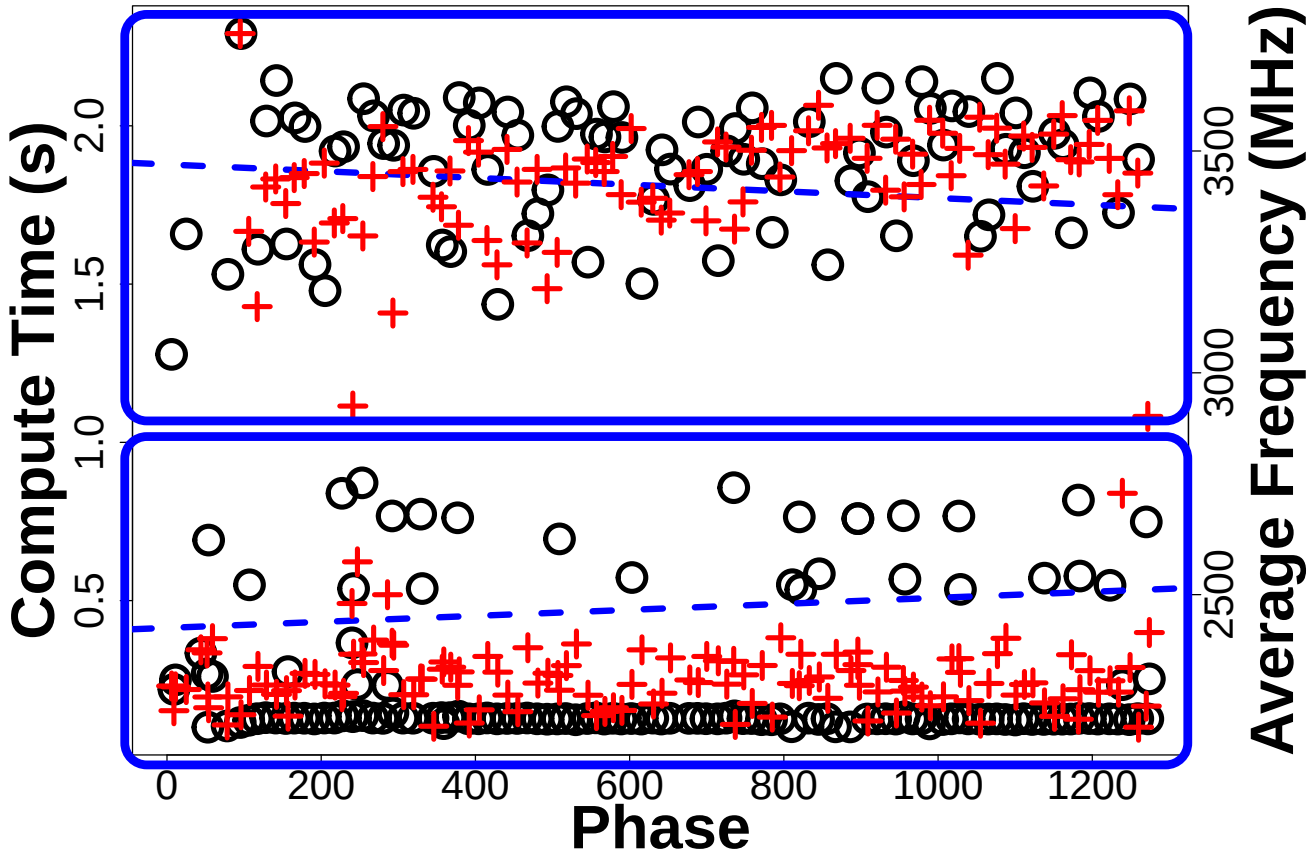

- Bimodal distribution of critical path times $< 1.0$s and $> 1.0$s
- Successive phases are similar, with only occasional jumps
- Average critical path frequency (Default) = **2507.4MHz**

# *ParaDis* critical path on 24 nodes (768 cores) - DVFS




- Average critical path frequency (Default) = **2467.3MHz**

# *ParaDis* critical path on 24 nodes (768 cores) - DDCM




- Very low frequency on non-critical cores for prolonged periods **reduces variation**, and **increases available thermal headroom** for critical cores
- Average critical path frequency (Default) = **2784.8MHz**

# Mitigating workload balance

## average results across all experiments

| Policy | %Power reduced | %Energy saved | %Time increase | Temp decrease (C) |
|---|---|---|---|---|
| DDCM | 19.3 | 15.1 | 5.3 | 3.2 |
| DVFS | 20.5 | 20.2 | 0.5 | 3.3 |
| Combined | 24.9 | 22.6 | 2.9 | 4.2 |

- ACR demonstrates that **dynamic control of power at runtime** is possible

# Mitigating workload balance

## average results across all experiments

| Policy | %Power reduced | %Energy saved | %Time increase | Temp decrease (C) |
|---|---|---|---|---|
| DDCM | 19.3 | 15.1 | 5.3 | 3.2 |
| DVFS | 20.5 | 20.2 | 0.5 | 3.3 |
| Combined | 24.9 | 22.6 | 2.9 | 4.2 |

- ACR demonstrates that **dynamic control of power at runtime** is possible
- At **Exascale**, runtimes such as ACR will allow
  - **more work to be run** at one time by using less power
  - **individual applications** to run faster by allowing a higher thermal headroom on critical cores

# Mitigating workload balance

## average results across all experiments

| Policy | %Power reduced | %Energy saved | %Time increase | Temp decrease (C) |
|---|---|---|---|---|
| DDCM | 19.3 | 15.1 | 5.3 | 3.2 |
| DVFS | 20.5 | 20.2 | 0.5 | 3.3 |
| Combined | 24.9 | 22.6 | 2.9 | 4.2 |

- ACR demonstrates that **dynamic control of power at runtime** is possible
- At **Exascale**, runtimes such as ACR will allow
  - **more work to be run** at one time by using less power
  - **individual applications** to run faster by allowing a higher thermal headroom on critical cores
- Energy optimization can also be performance optimization

# Saving energy in memory-bound applications

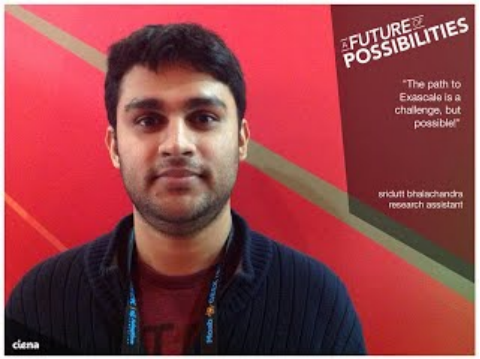
- Many HPC applications are memory-bound
- Memory operations are seldom visible to OS/runtime
  - Power wasted in CPU while waiting on memory
- Approach
  - sample table of request occupancy in memory subsystem
  - use DVFS to slow non-critical cores

## Published work

1. Improving Energy Efficiency in Memory-constrained Applications Using Core-specific Power Control (E2SC 2017)

# Acknowledgements

Ph.D. research of Sridutt Bhalachandra (Argonne National Labs Exascale Group)




## Published work

1. An Adaptive Core-specific Runtime for Energy Efficiency (IPDPS 2017)
2. Using Dynamic Duty Cycle Modulation to improve energy efficiency in High Performance Computing (HPPAC 2015)