Power and Energy in High-Performance Computing

- A similar story as with resilience
  - Exascale systems will have extremely large numbers of cores
  - Power limit for exascale system is (approximately) 20 Megawatts
  - Today's petascale (roughly, 10-30 PF) consume slightly less than 10 MW
  - So, "all" we need to do is improve performance by a factor of 100 while restricting power increase to a power of 2

Taking a step back in time…

- Interest in power and energy (mostly energy) started in the mobile computing community
  - Late 1990s
  - Focused on battery life
  - Mobile computing has a huge market (unlike HPC, which is in many ways a niche market)

DVFS

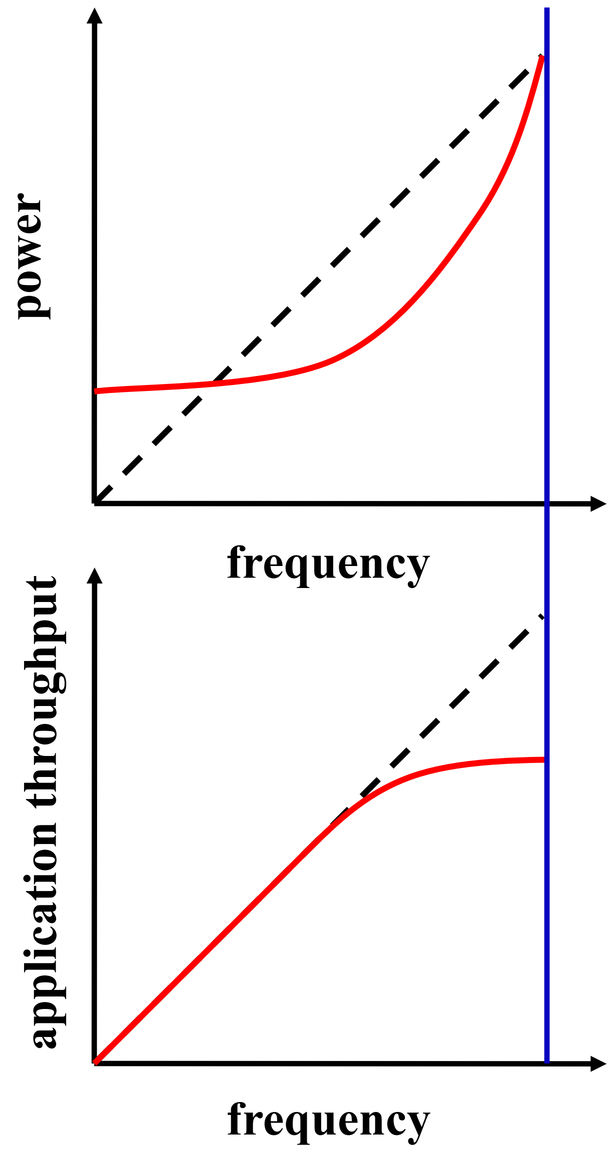
- First major feature for saving power and energy was dynamic voltage and frequency scaling (DVFS)
  - Execute program at lower frequency/voltage
  - Idea existed in a sense much earlier (overclocking)
  - Power can be greatly reduced via DVFS
- Idea was that users of mobile devices rarely use the processor to its full capacity
  - Think about what you do on your laptop
    - Memory bound activities
    - Network bound activities

DVFS, pictorially

- Reduce frequency & voltage
  - Reduces CPU power & performance
  - Energy-time tradeoff

- Why is this a good idea?
  - Applications may not be CPU-bound
  - CPU is large power consumer

power $\propto$ frequency x voltage^2



Three stages in the evolution of high-performance, power-aware computing

1. Lower energy-delay product of HPC jobs
   – i.e., some time delay is acceptable for lower energy
2. Lower energy without a time increase
3. Optimize performance under a fixed power budget

Stage 1: Lower Energy of HPC Jobs

- Coincided with desire to reduce energy in society
- Observation: parallel programs are inefficient
  - Recall: Parallel efficiency is the ratio of speedup to the number of cores
  - Parallel efficiency falls in range (0,1); 1 is best
  - Gordon Bell prize given at Supercomputing conference each year for top performing HPC app
    - Typically, the parallel efficiency of winner is between 50% and 80%---and that's the winner!

What does poor parallel efficiency mean?

- Reasons why parallel efficiency is poor:
  - Communication/synchronization
  - Load imbalance
  - Purely sequential phases
- So, why should we run fast?
  - Essentially, blocking communication is an opportunity to use DVFS to lower CPU speed and therefore CPU power (and therefore energy)
- Goal: save a lot of energy and increase execution time only slightly (if at all)

Other reason to slow down CPU

- Memory bottleneck
  - If program is spending a lot of time accessing memory, the CPU speed is (relatively) irrelevant
  - Another opportunity to save energy with only a modest increase in execution time
  - Note: not all cache levels run at chip speed

$T(f_r) = (T_{cpu} * f_{base}/f_r) + T_{mem}$

  - f_{base} is top frequency; f_r is reduced frequency
  - T_{cpu} (T_{mem}) is time spent in CPU (memory) ops
  - Unfortunately, determining T_{cpu} and T_{mem} is not simple (depends on hardware and program)

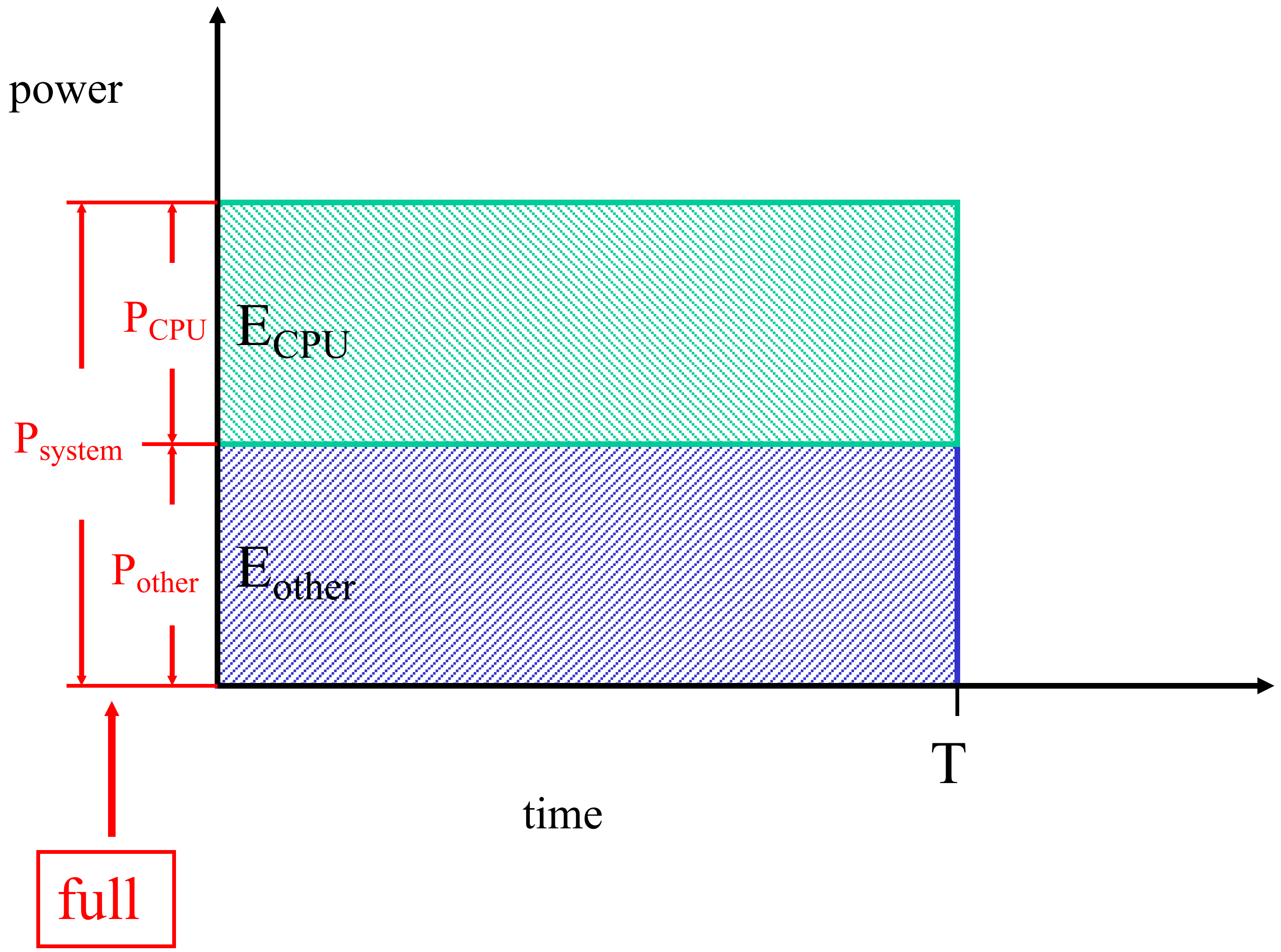
How does change in frequency on a core affect execution time?

- Complex: depends on mix of instructions
  - Memory bound vs CPU bound (or in the middle)
- Many have studied this problem
  - Mostly architects: create new architectural features that allow for a more accurate prediction

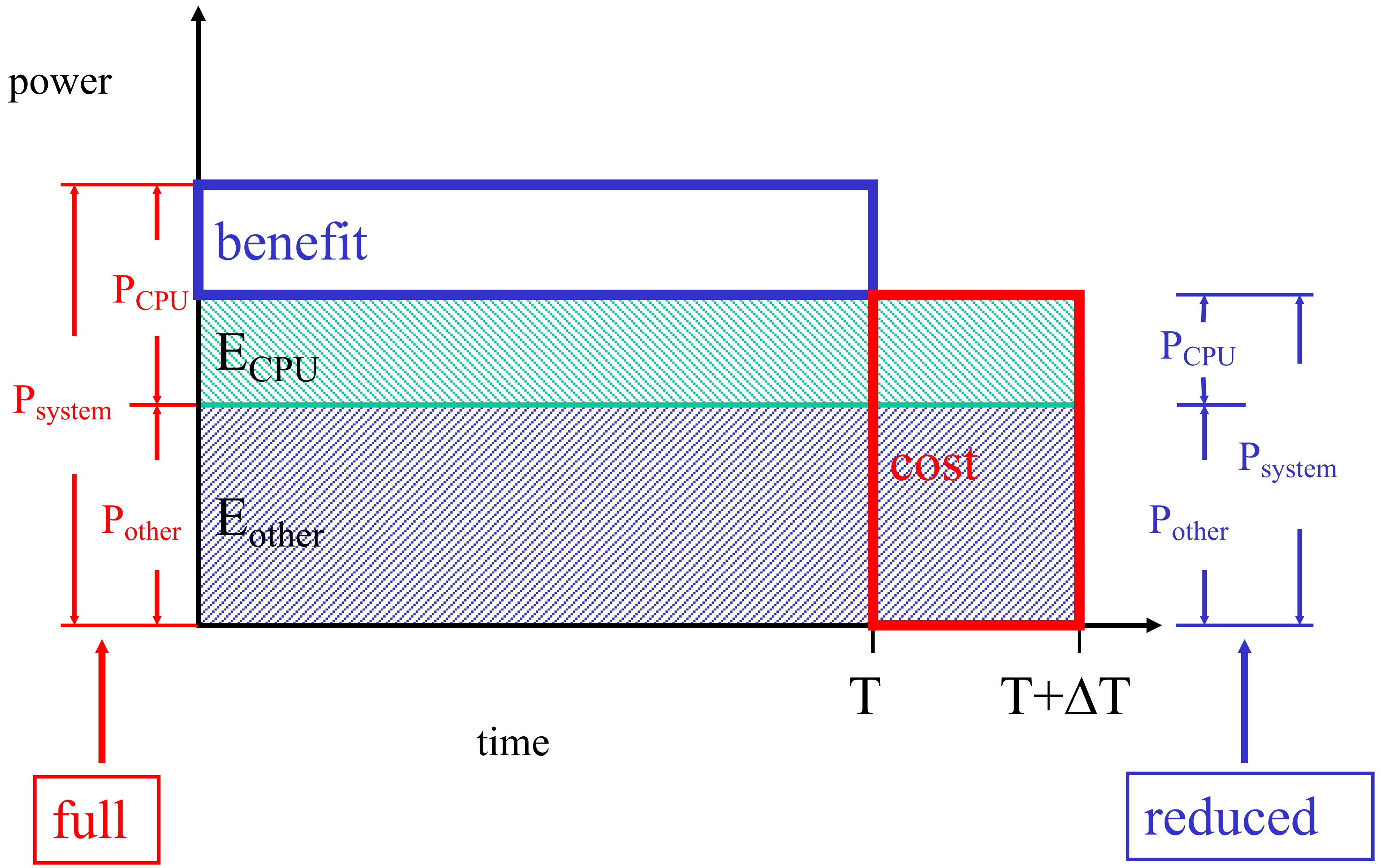
How might we decide what’s good?

- It’s a two-dimensional problem: energy and time
- Many (bad) metrics for evaluation discussed
  - Energy * Delay
  - Energy * Delay^2
  - Really, it depends on who you ask

Is DVFS a win?



Is DVFS a win?


power
benefit
P_{CPU}
E_{CPU}
P_{system}
P_{other}
E_{other}
cost
P_{CPU}
P_{system}
P_{other}
T
T+ΔT
time
full
reduced

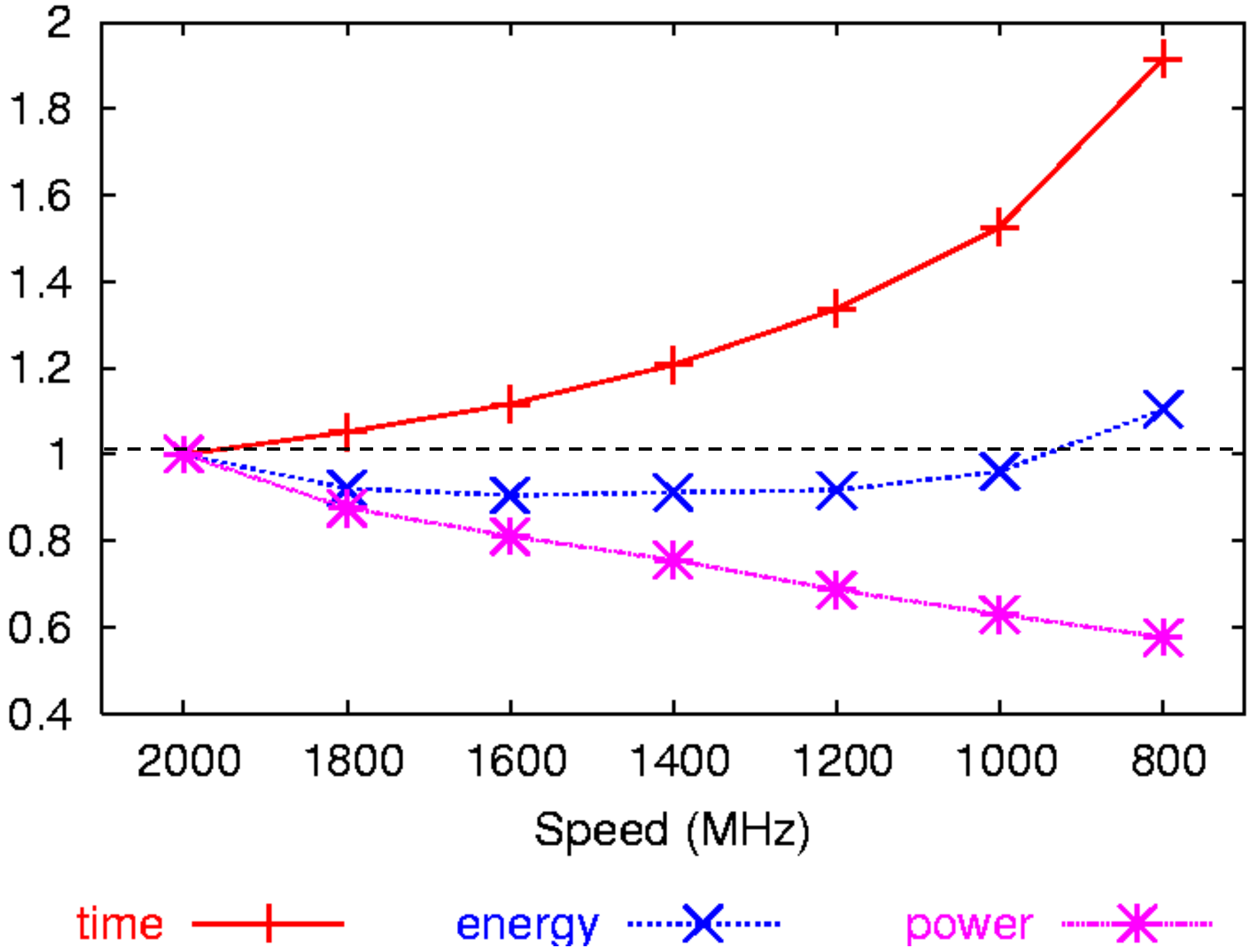
Some results

- Cluster used: 10 nodes, AMD Athlon-64
  - Processor supports 7 frequency-voltage settings

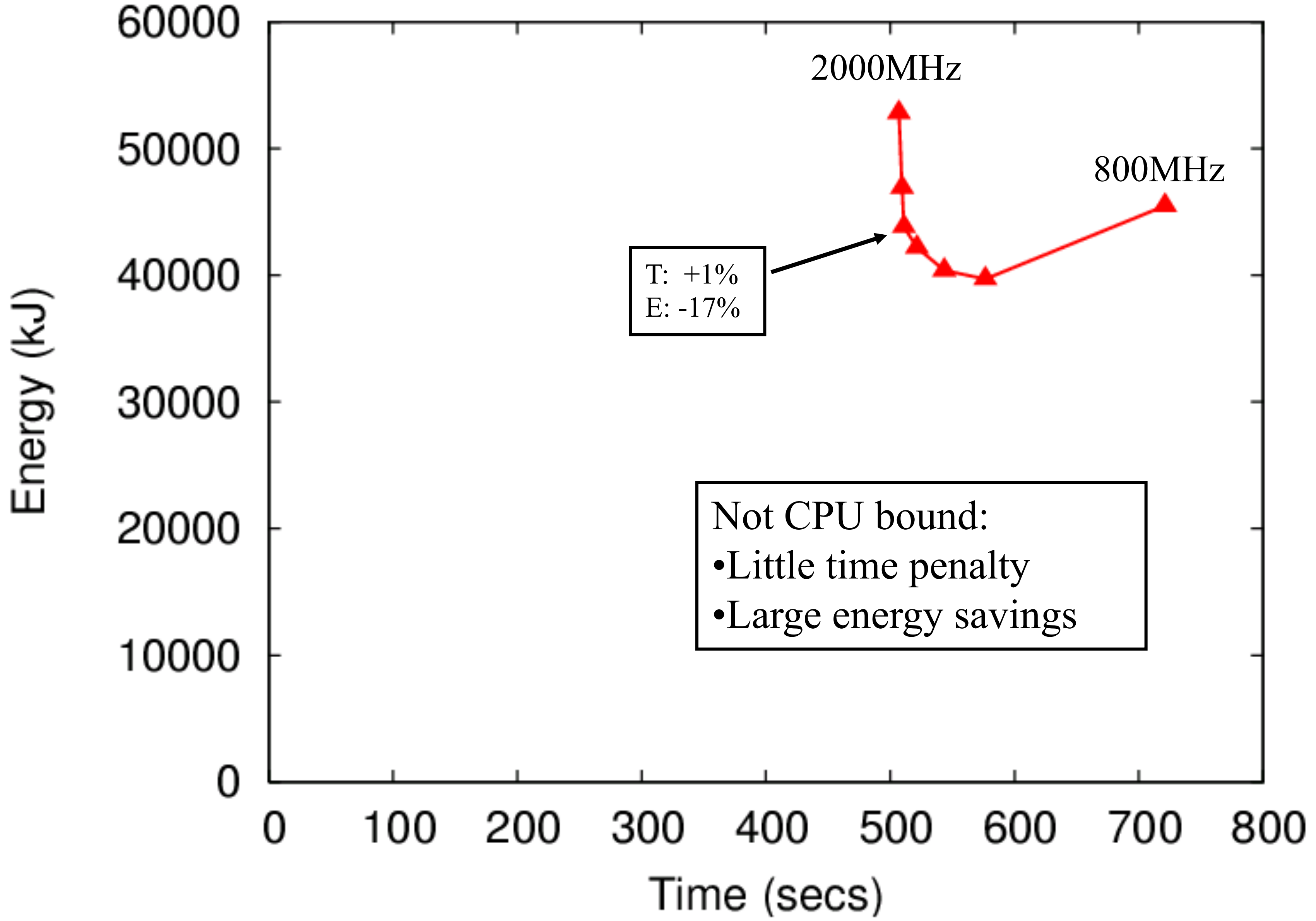
Frequency (MHz)	2000	1800	1600	1400	1200	1000	800
Voltage (V)	1.5	1.4	1.35	1.3	1.2	1.1	1.0

- Measure
  - Wall clock time (gettimeofday system call)
  - Energy (external power meter)

NAS Composite Results (1 node)



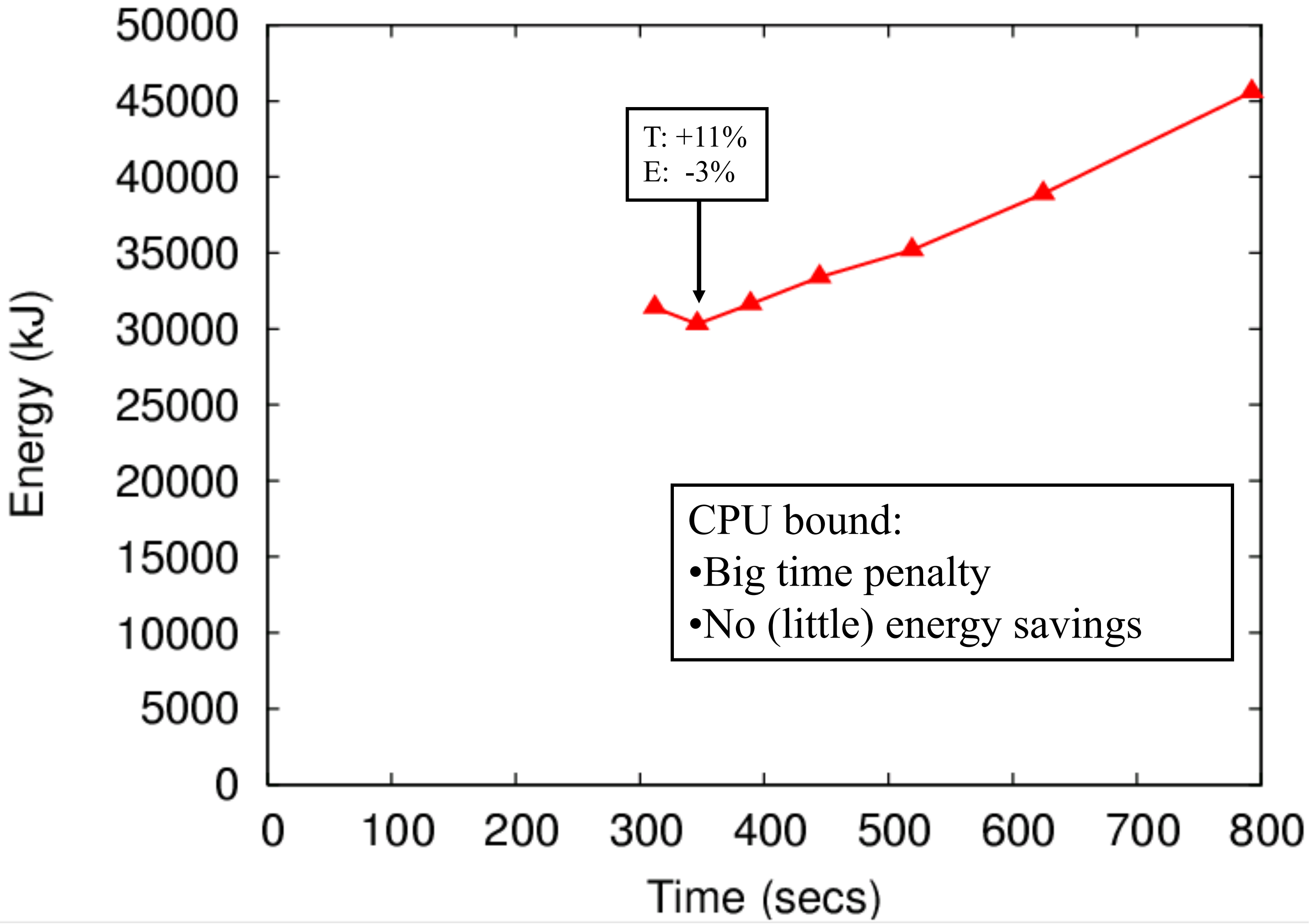
CG – 1 node



T: +1%
E: -17%

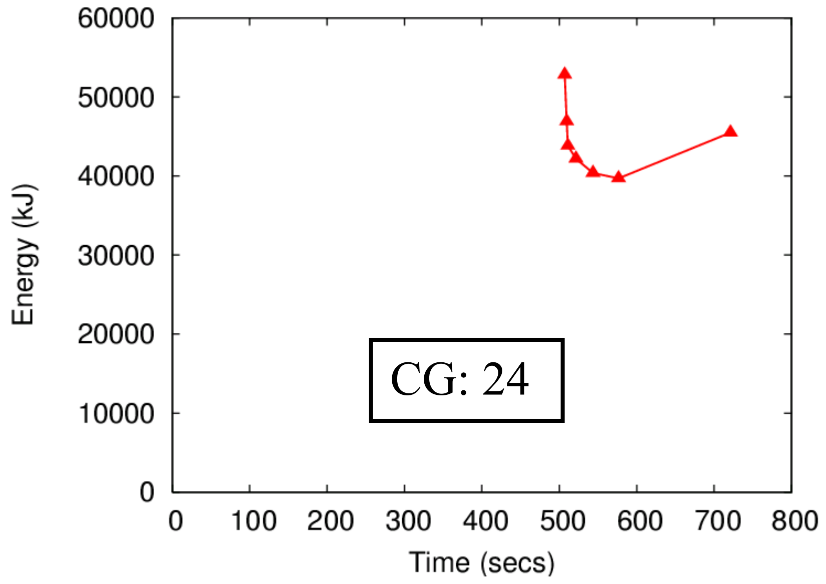
Not CPU bound:
- Little time penalty
- Large energy savings

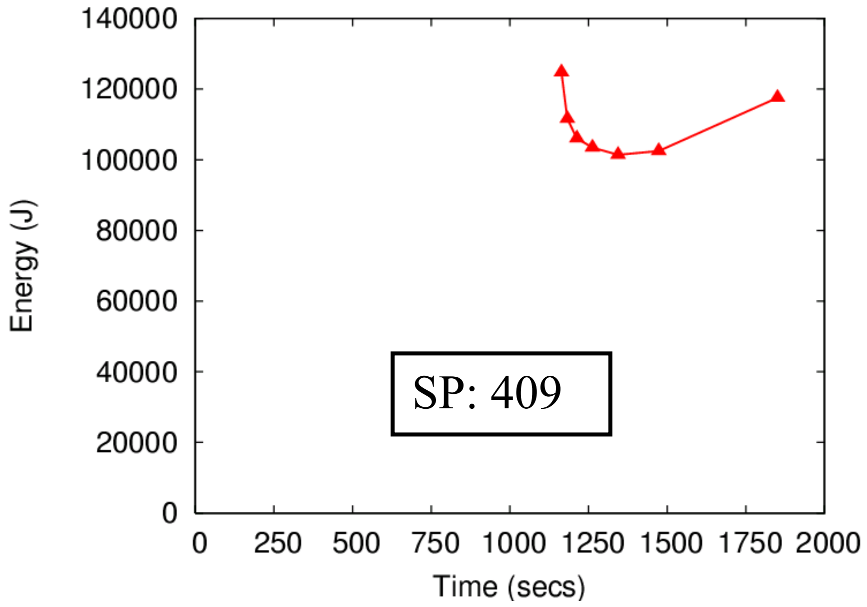
EP – 1 node

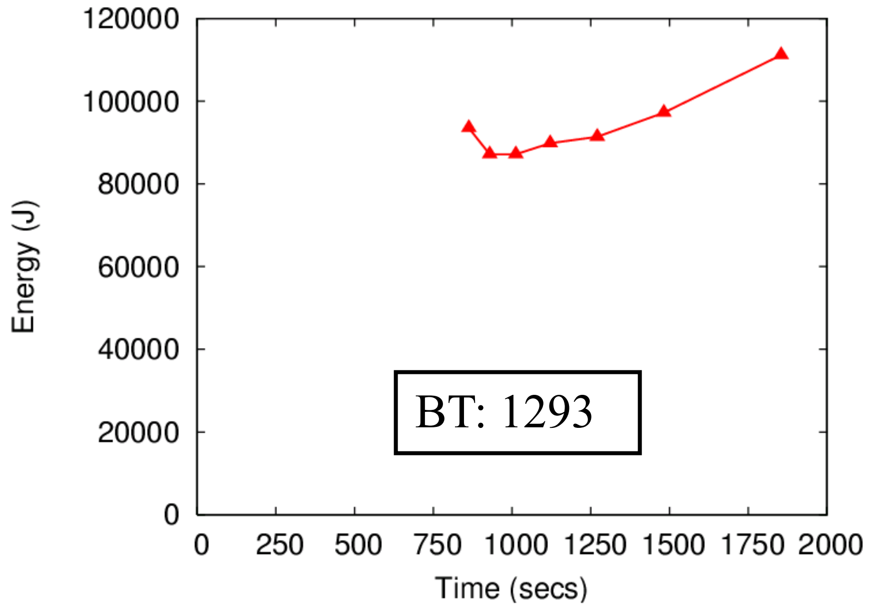


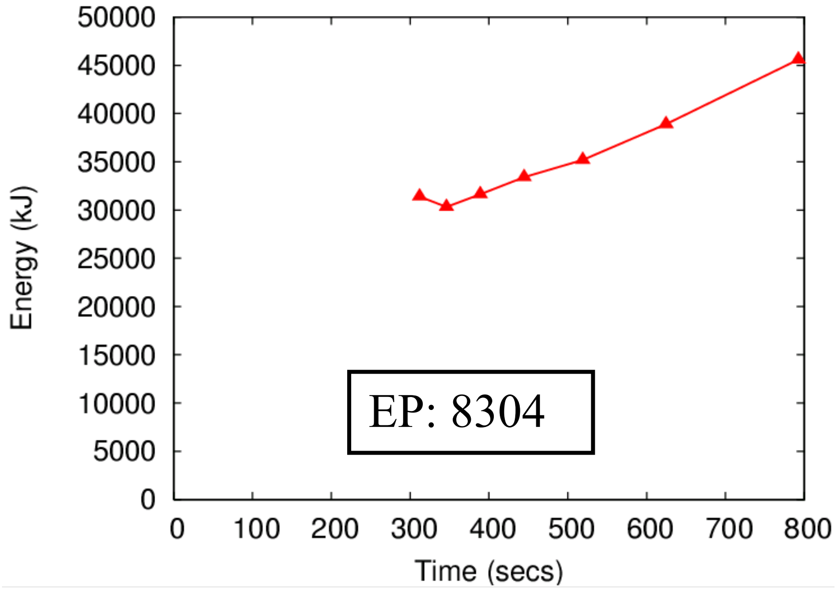
CPU bound:
•Big time penalty
•No (little) energy savings

Operations per memory access

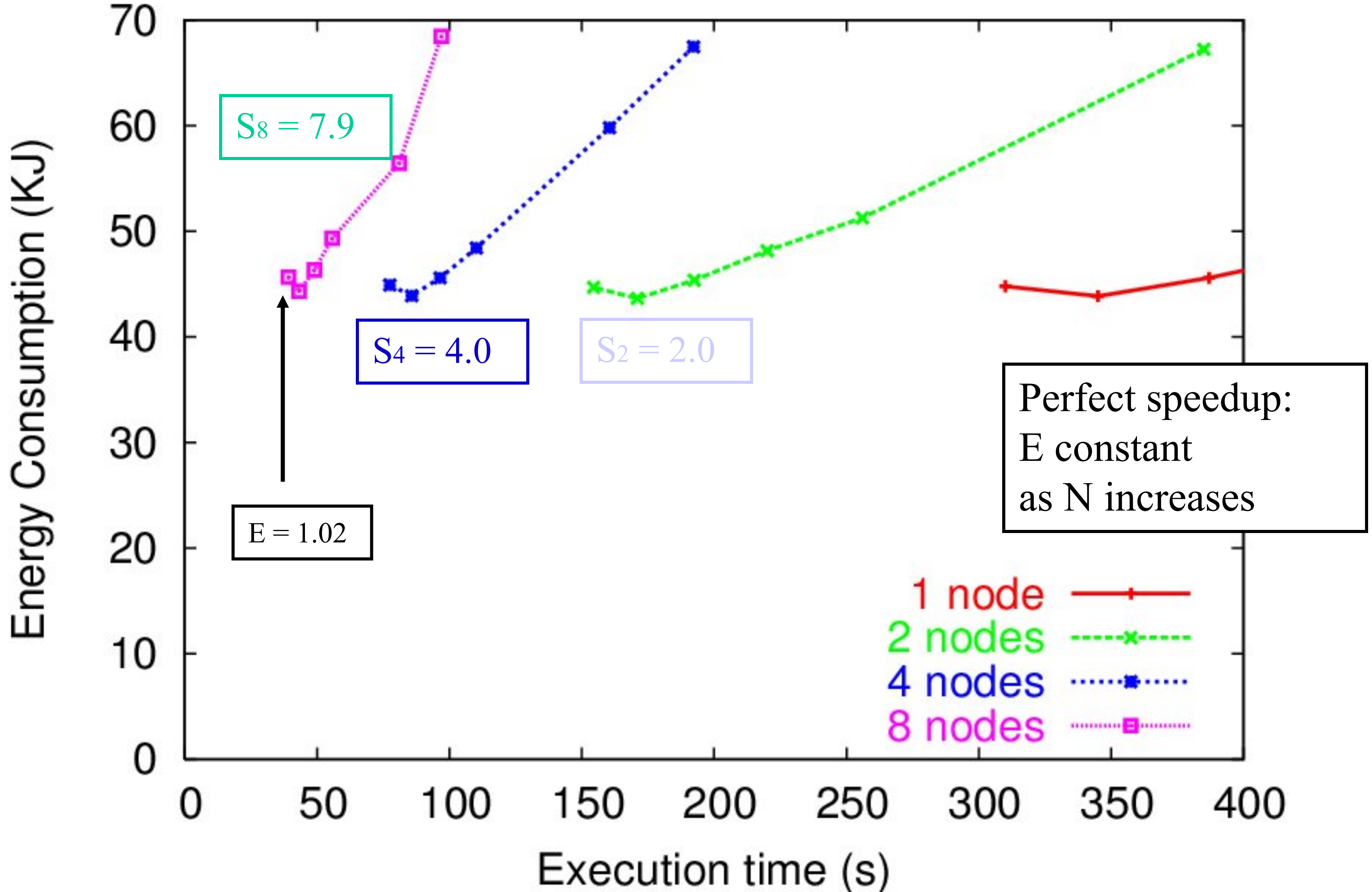






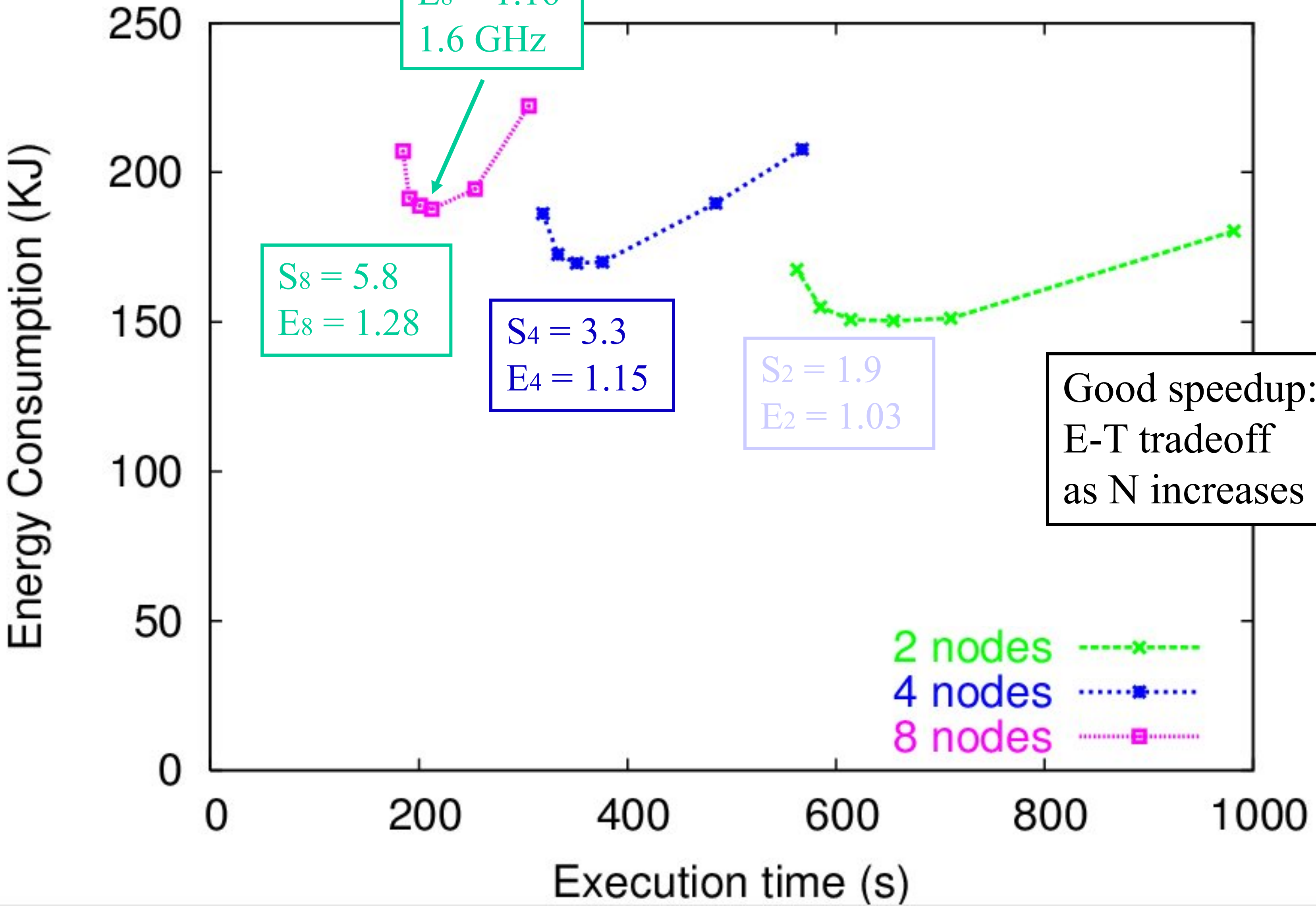


Multiple nodes – EP



Perfect speedup:
E constant
as N increases

Multiple nodes – LU



Good speedup:
E-T tradeoff
as N increases