Variation Aware Application Scheduling and Power Management for Chip Multiprocessors

Radu Teodorescu* and Josep Torrellas

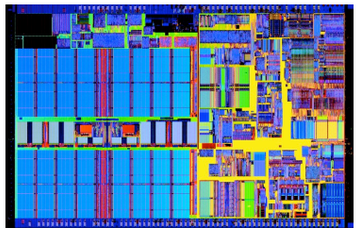
Computer Science Department
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

*now at Ohio State University

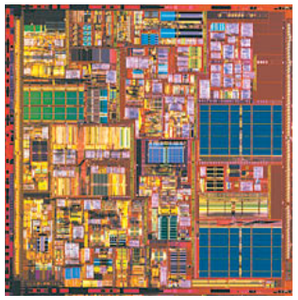
i-acoma group

Technology scaling continues

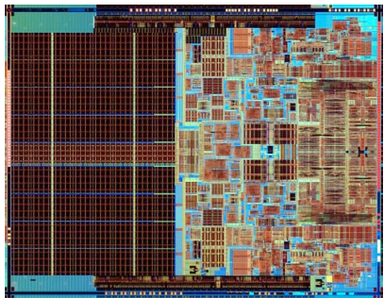
Pentium 3



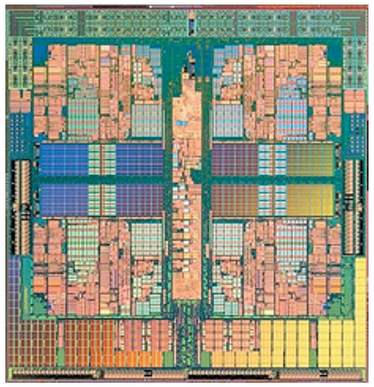
Pentium 4



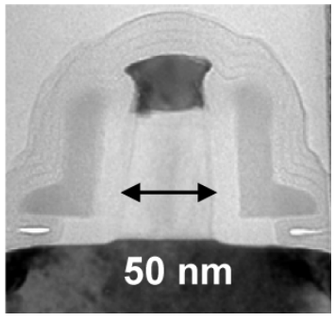
Core 2 Duo

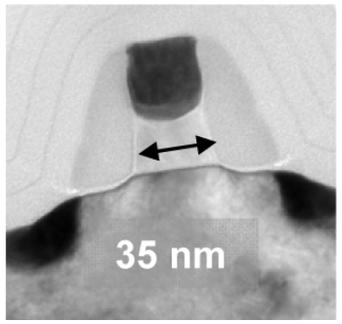


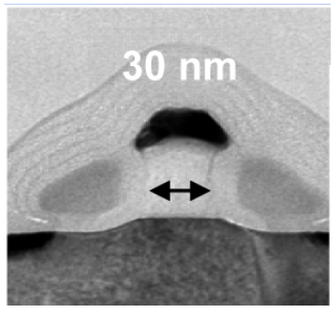
Quad Opteron

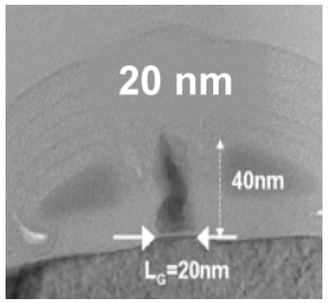


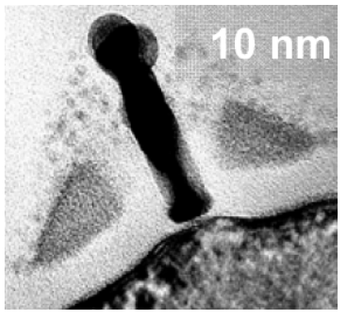
number
of transistors





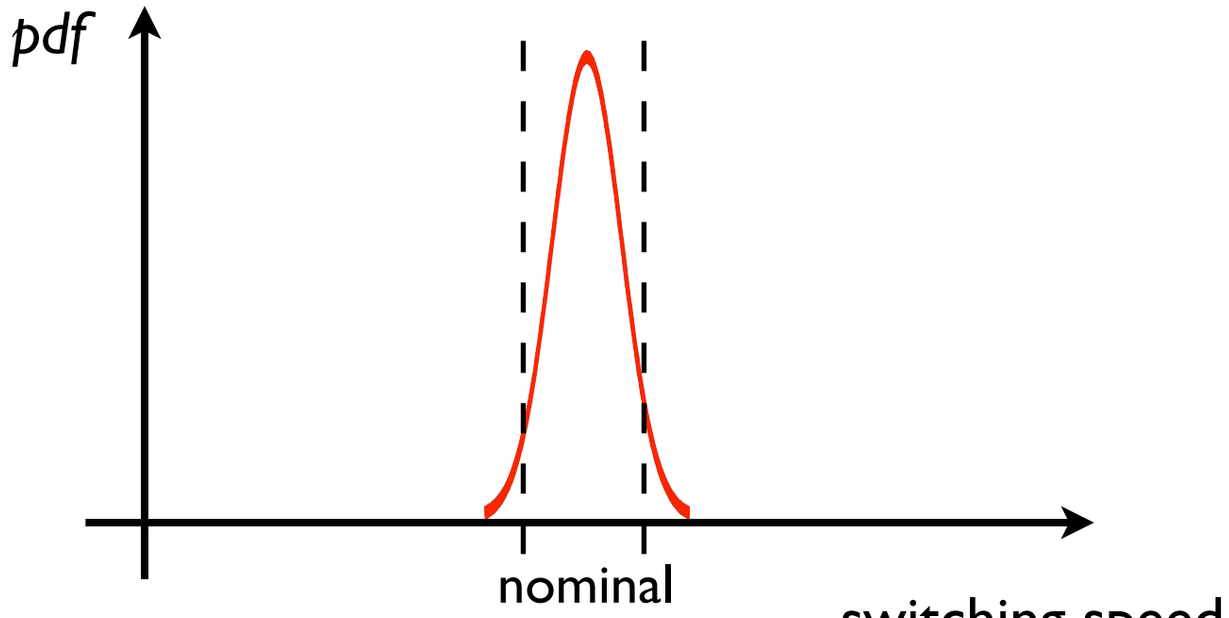




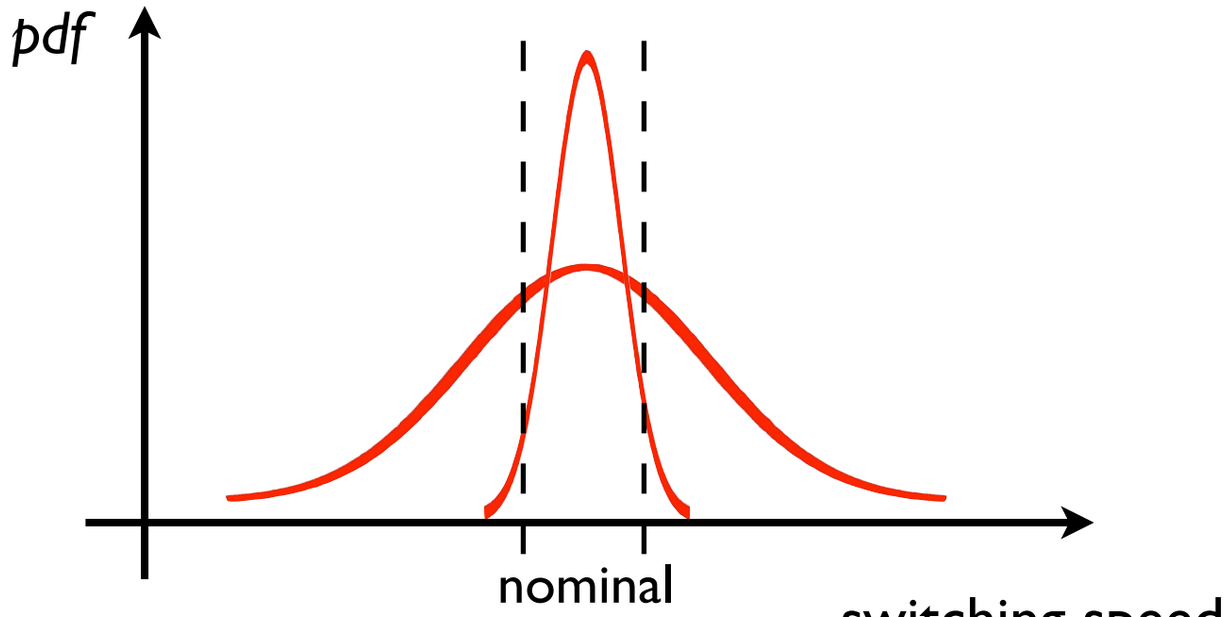


transistor
size

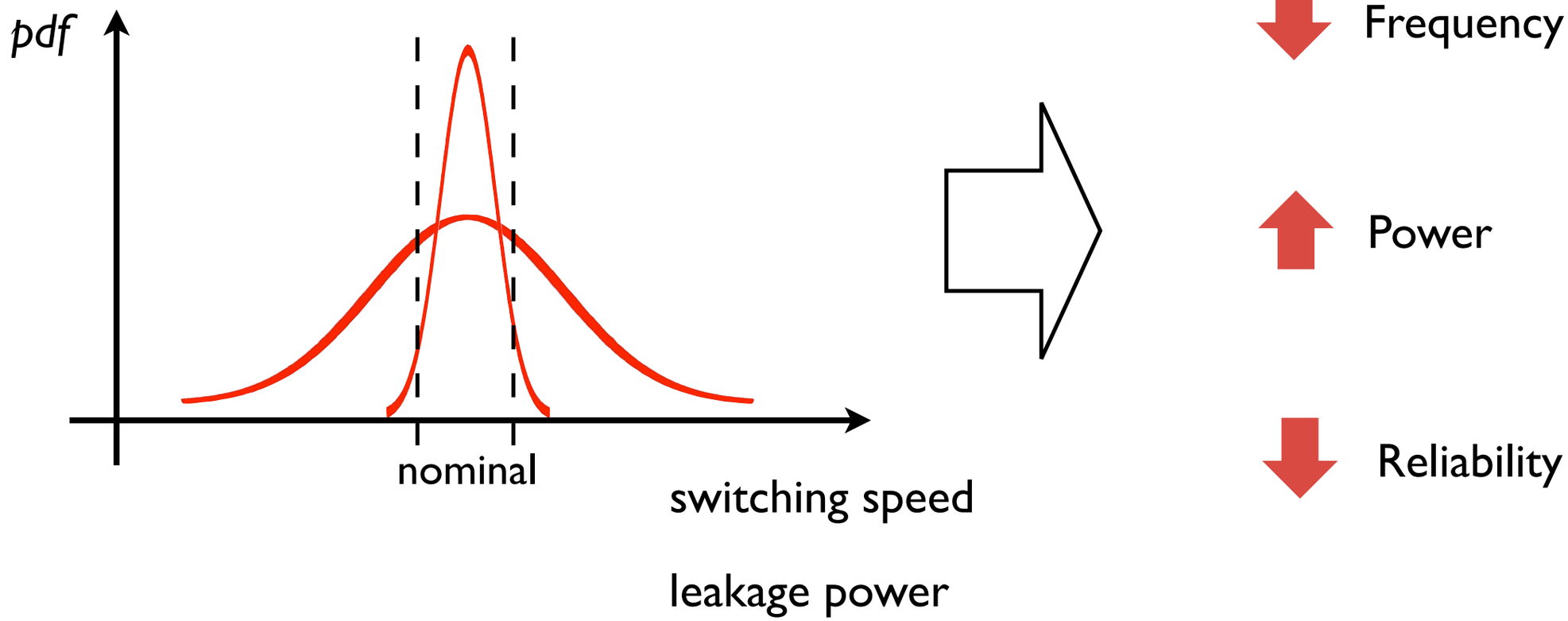
Variation in transistor parameters



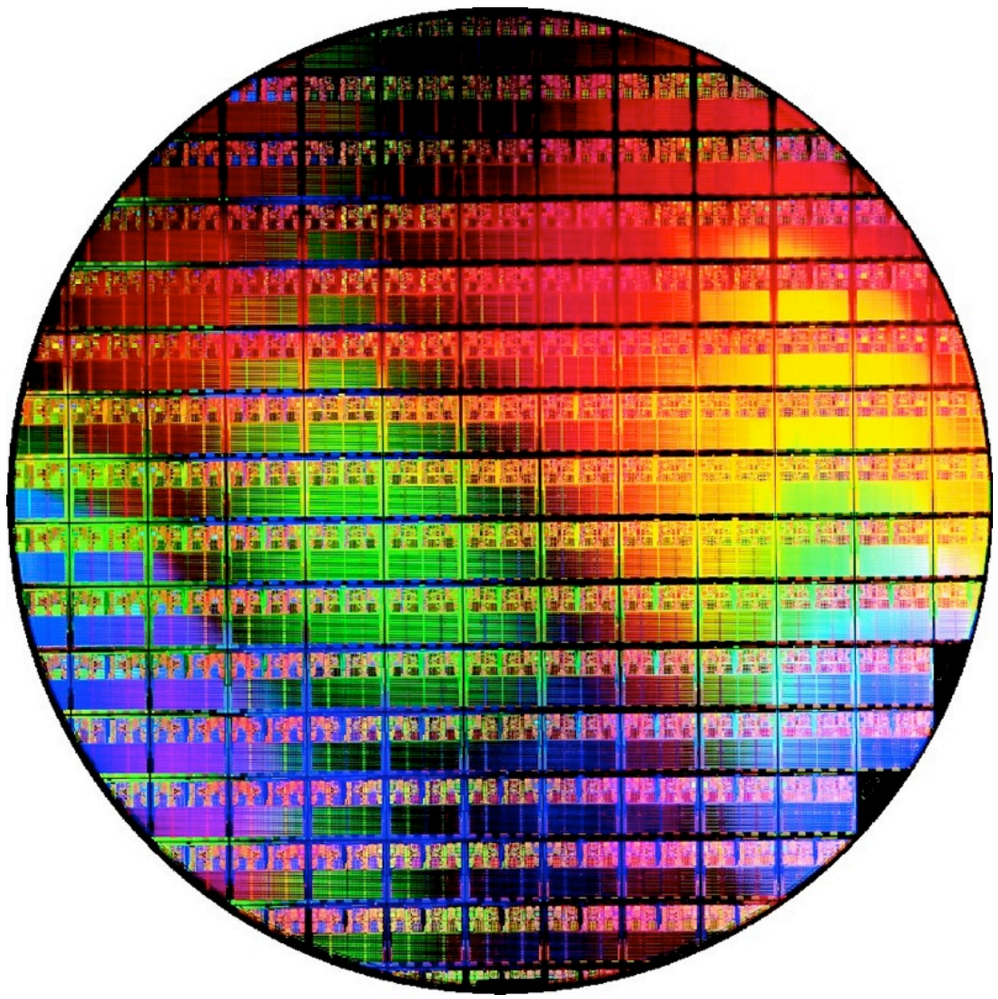
Variation in transistor parameters



Variation in transistor parameters

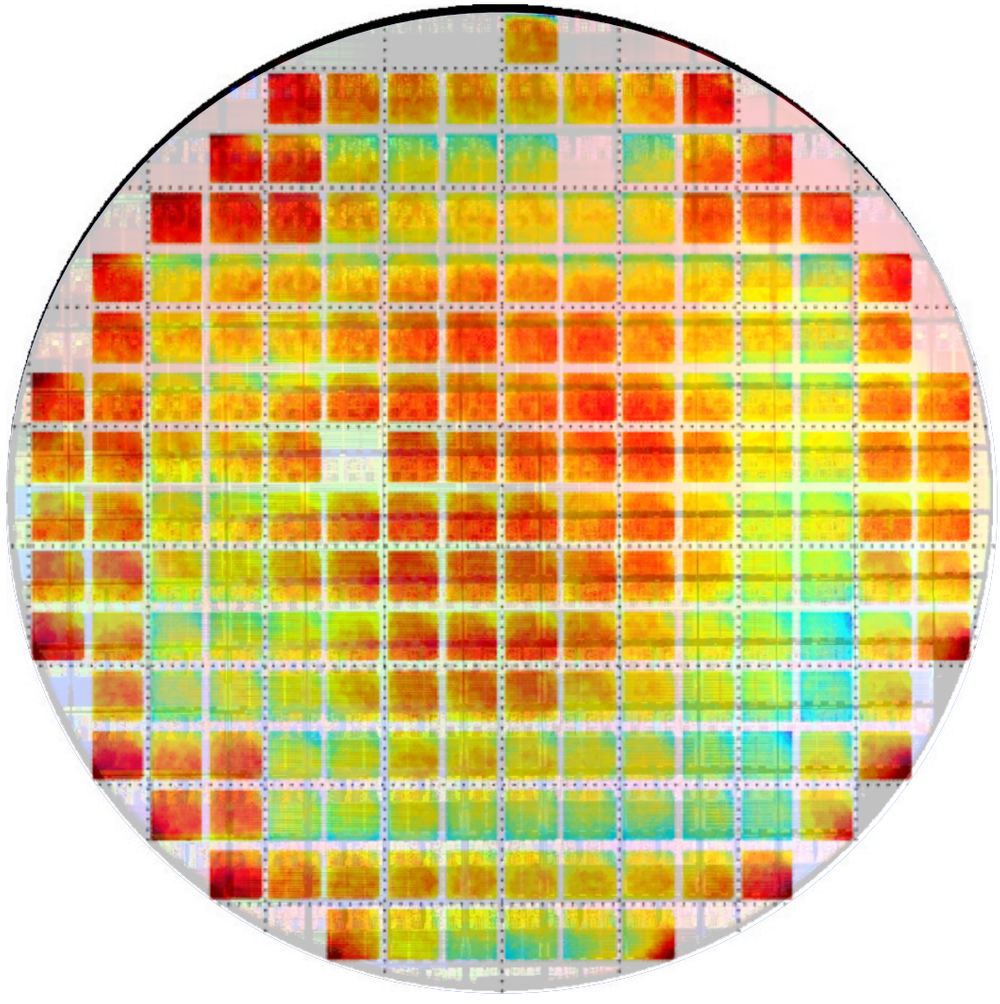


Variation components

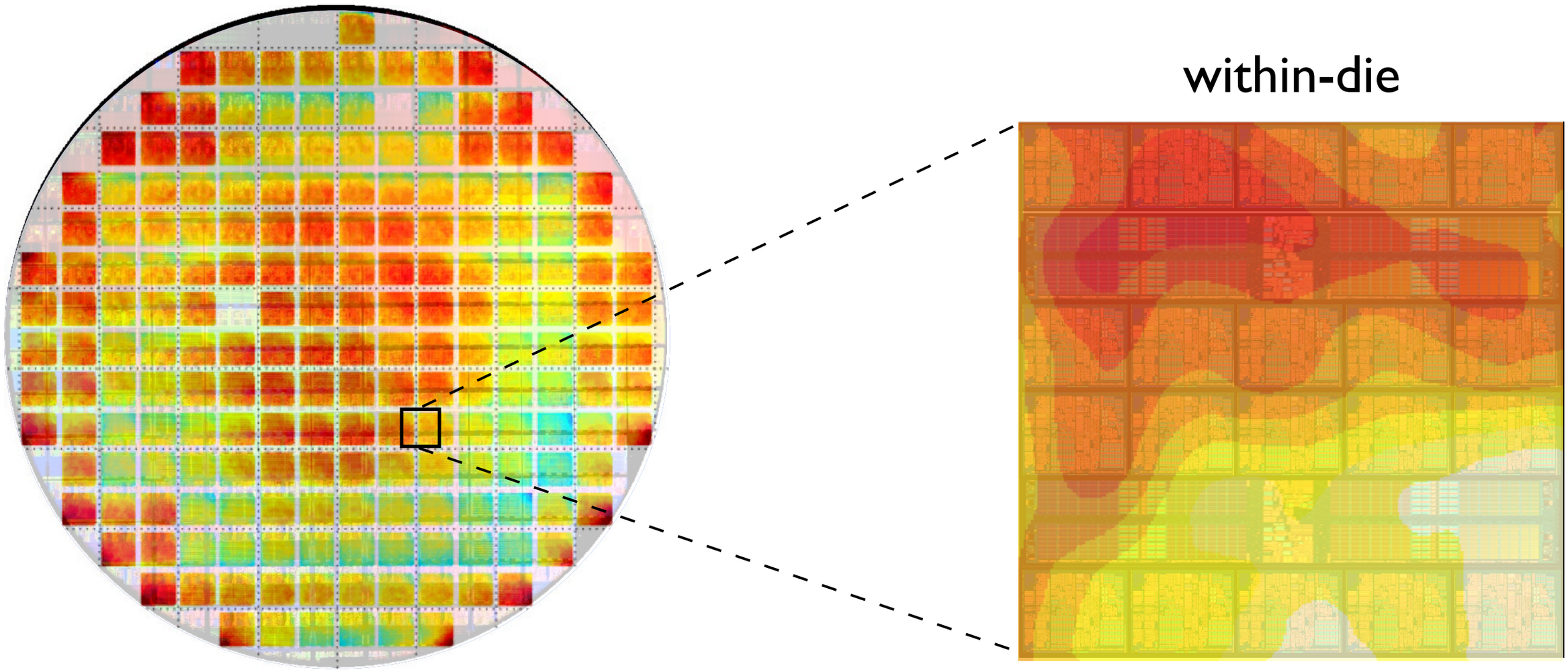


Variation components

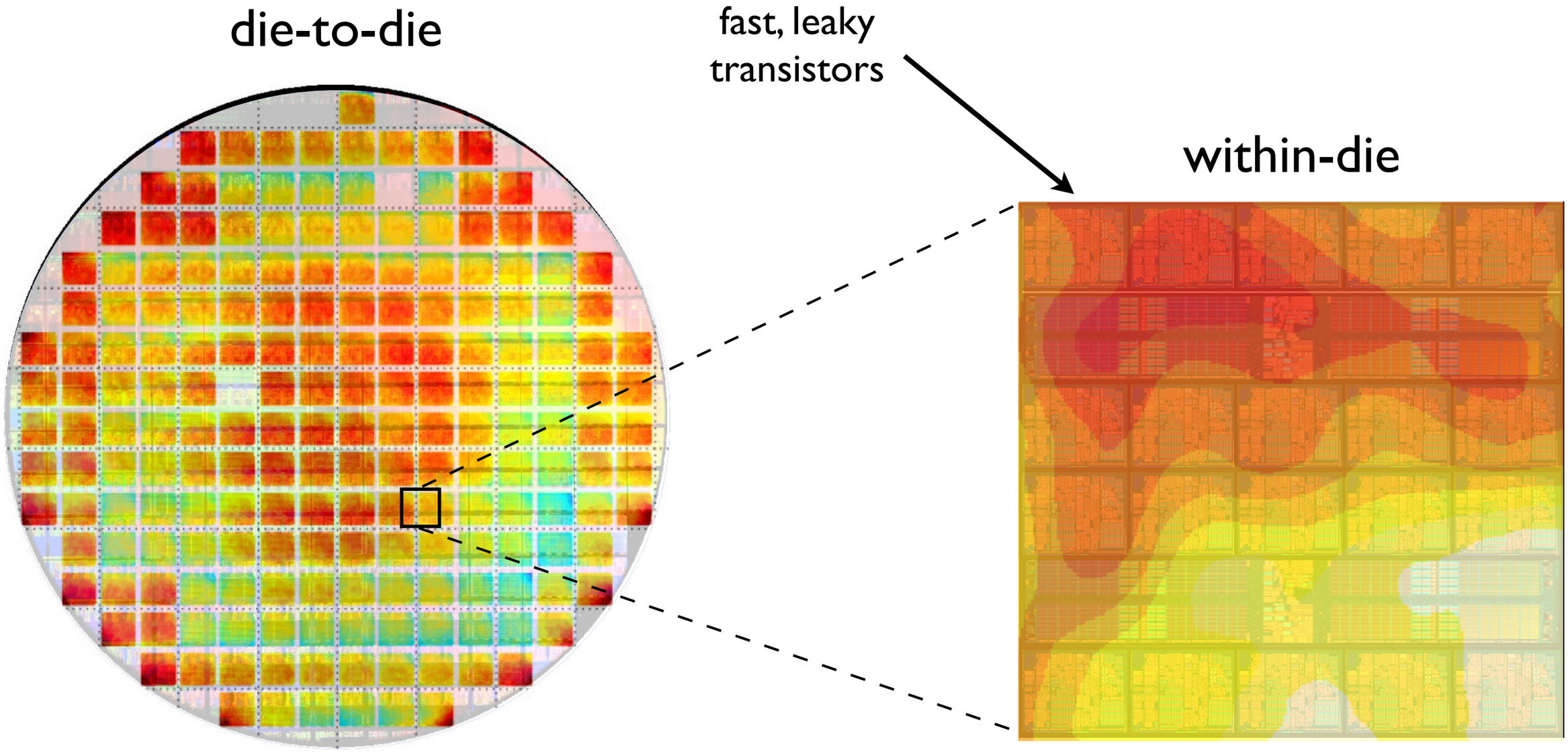
die-to-die



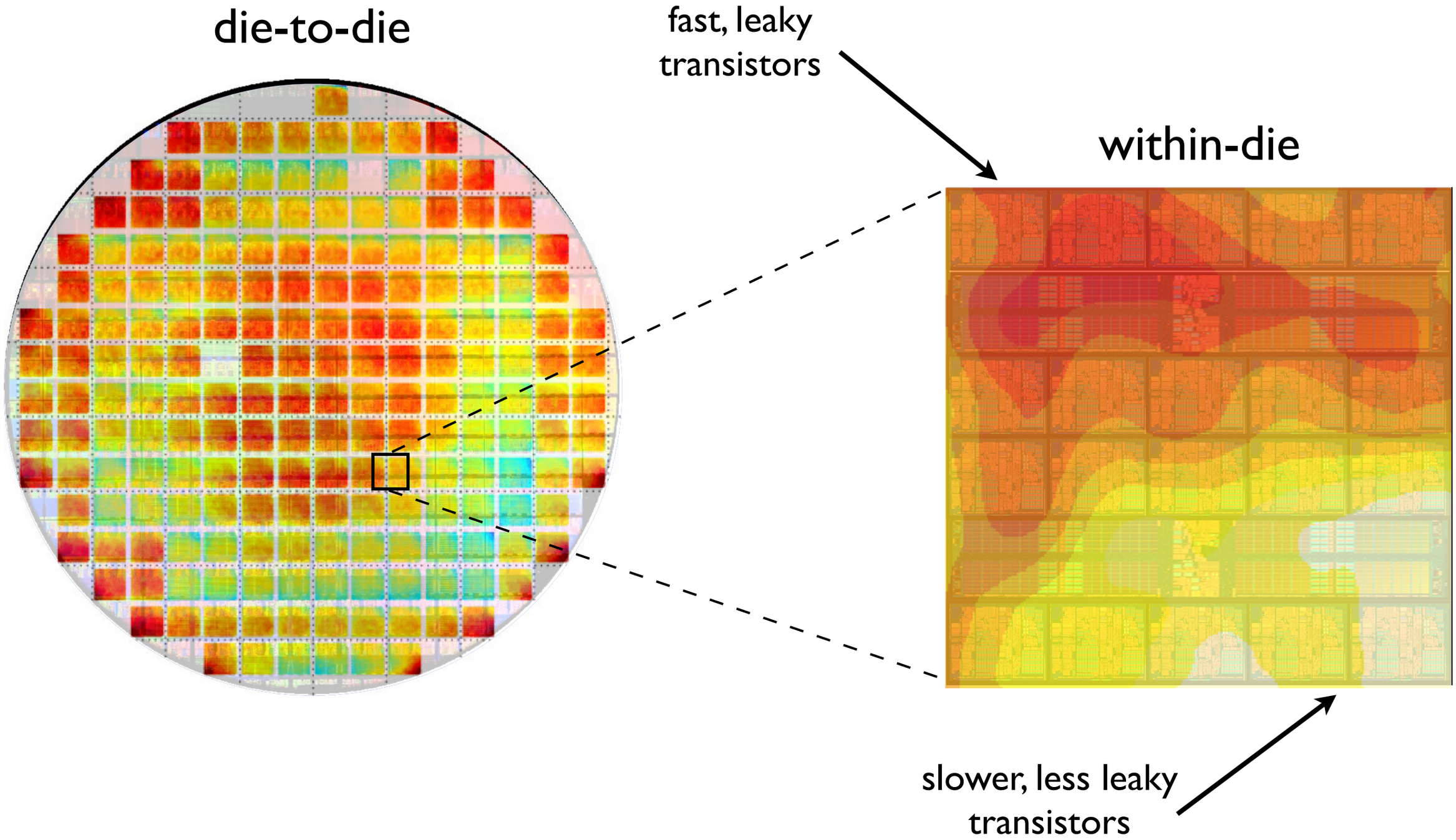
Variation components



Variation components

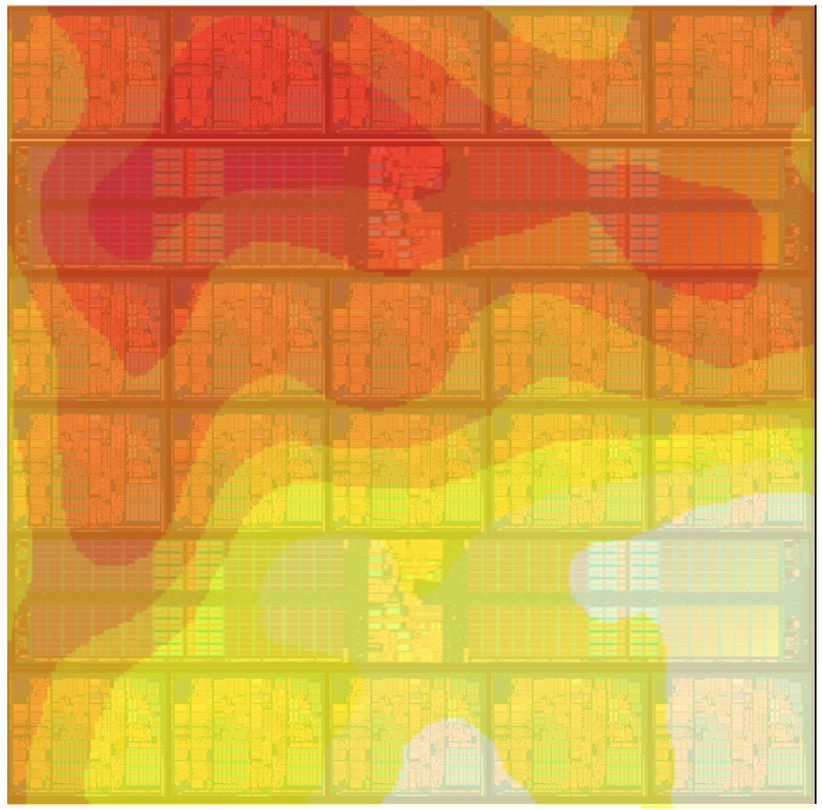


Variation components

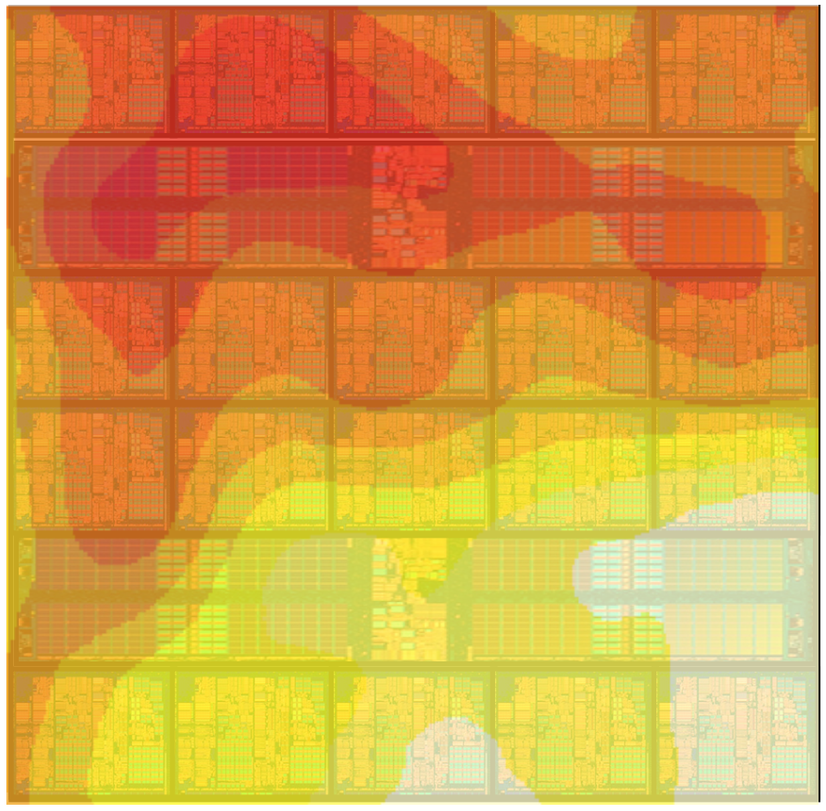


Variation components

within-die

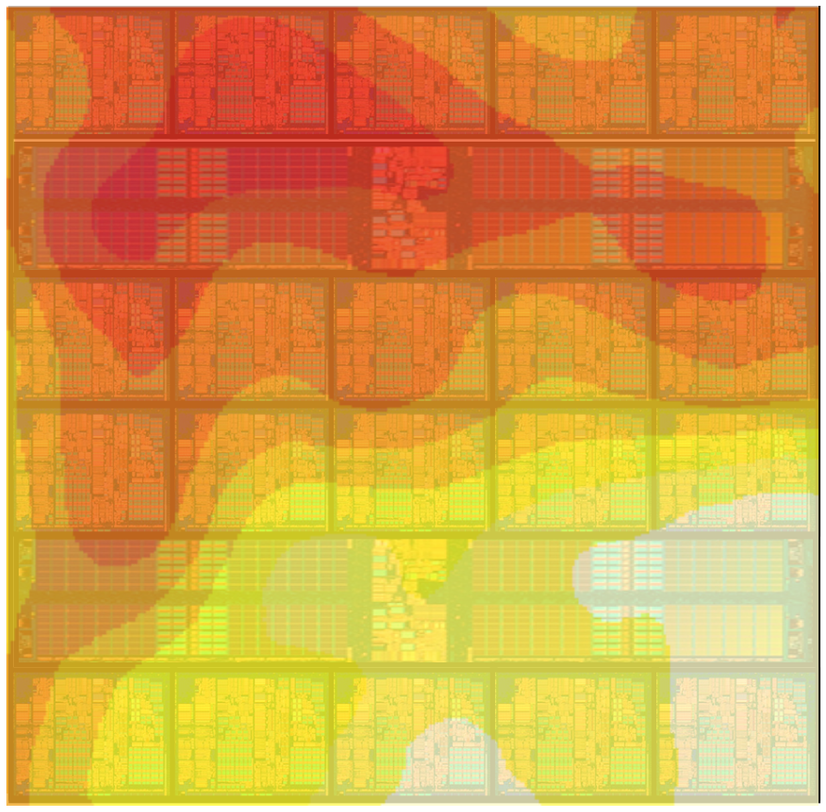


Effects of within-die variation



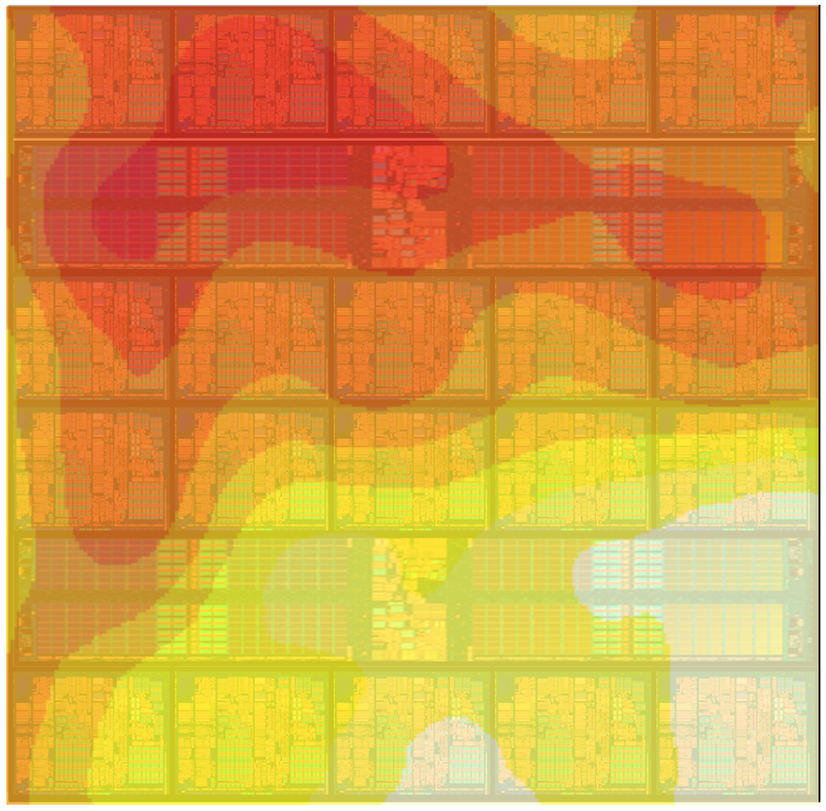
Effects of within-die variation

- CMPs: significant core-to-core variation in frequency and power



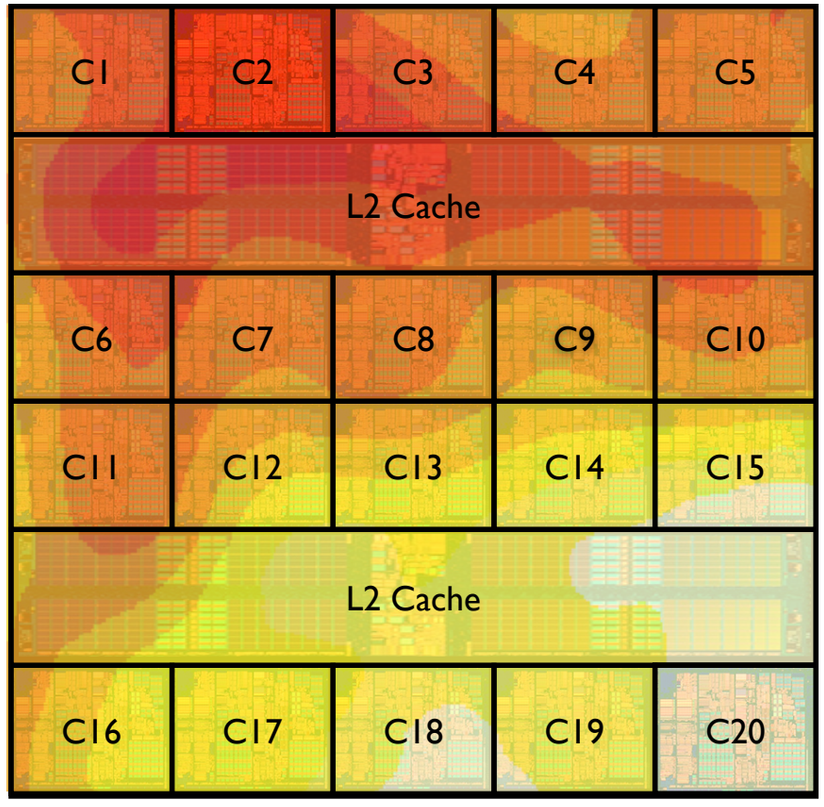
Effects of within-die variation

- CMPs: significant core-to-core variation in frequency and power
- We model a 20-core CMP, 32nm



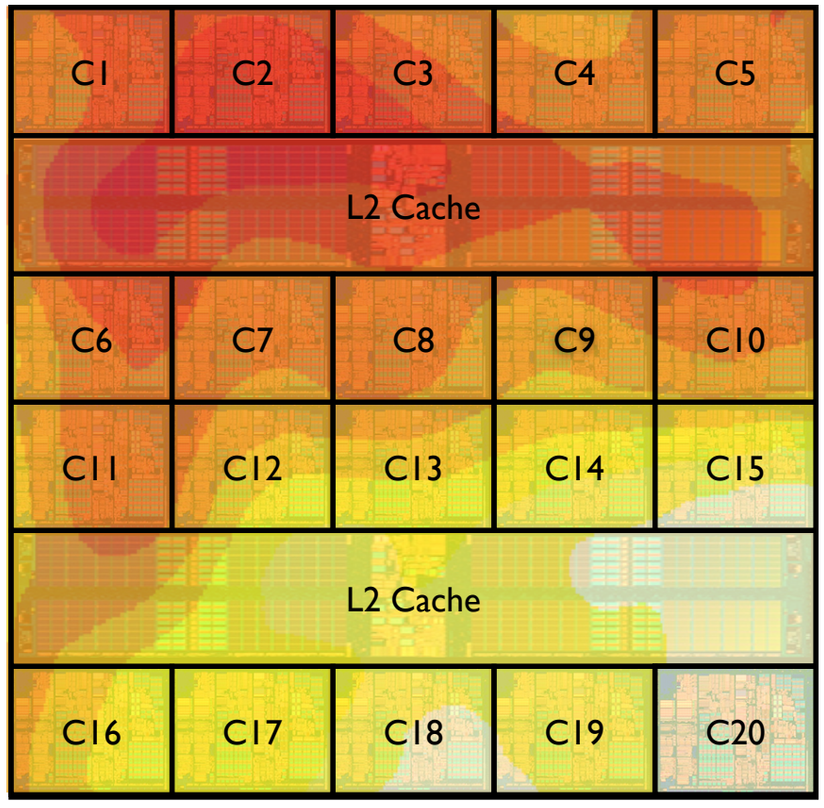
Effects of within-die variation

- CMPs: significant core-to-core variation in frequency and power
- We model a 20-core CMP, 32nm



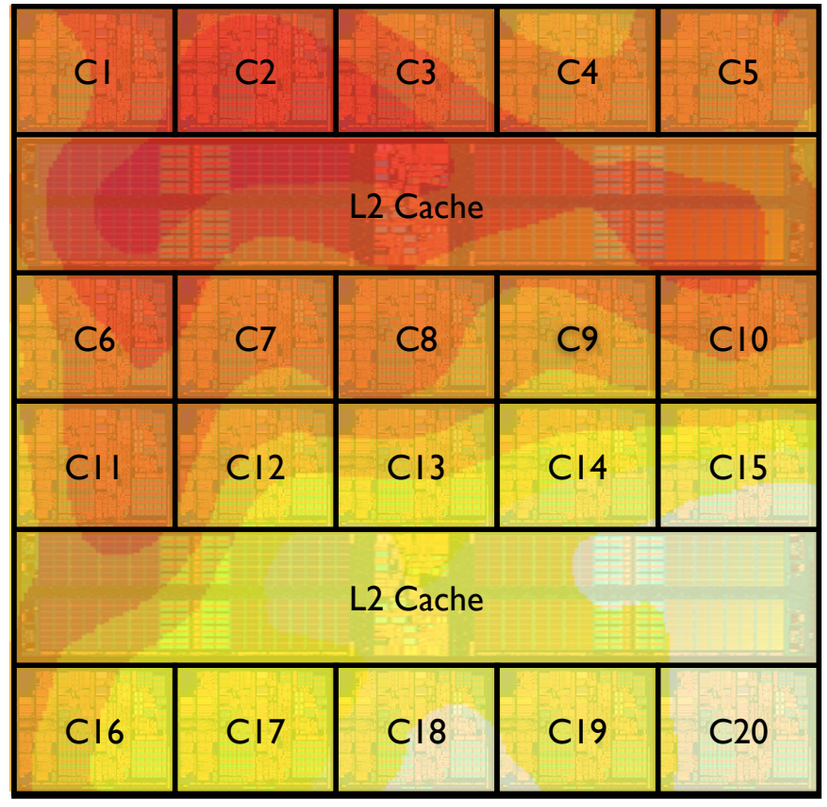
Effects of within-die variation

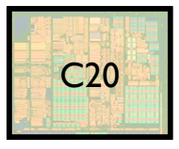
- CMPs: significant core-to-core variation in frequency and power
- We model a 20-core CMP, 32nm



Effects of within-die variation

- CMPs: significant core-to-core variation in frequency and power
- We model a 20-core CMP, 32nm





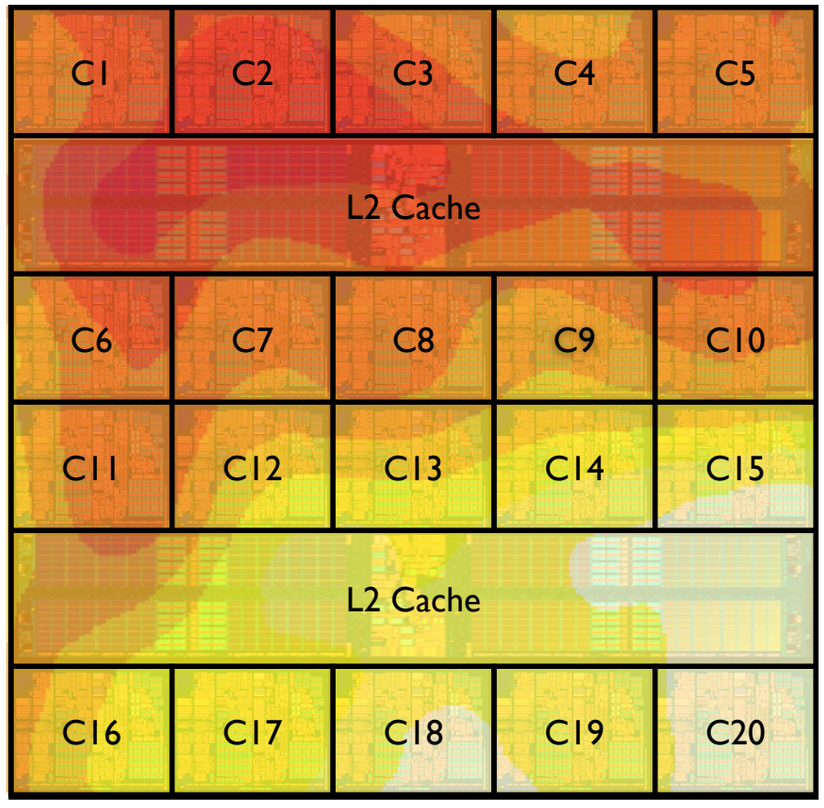
Effects of within-die variation

- CMPs: significant core-to-core variation in frequency and power
- We model a 20-core CMP, 32nm

fastest

vs.

slowest



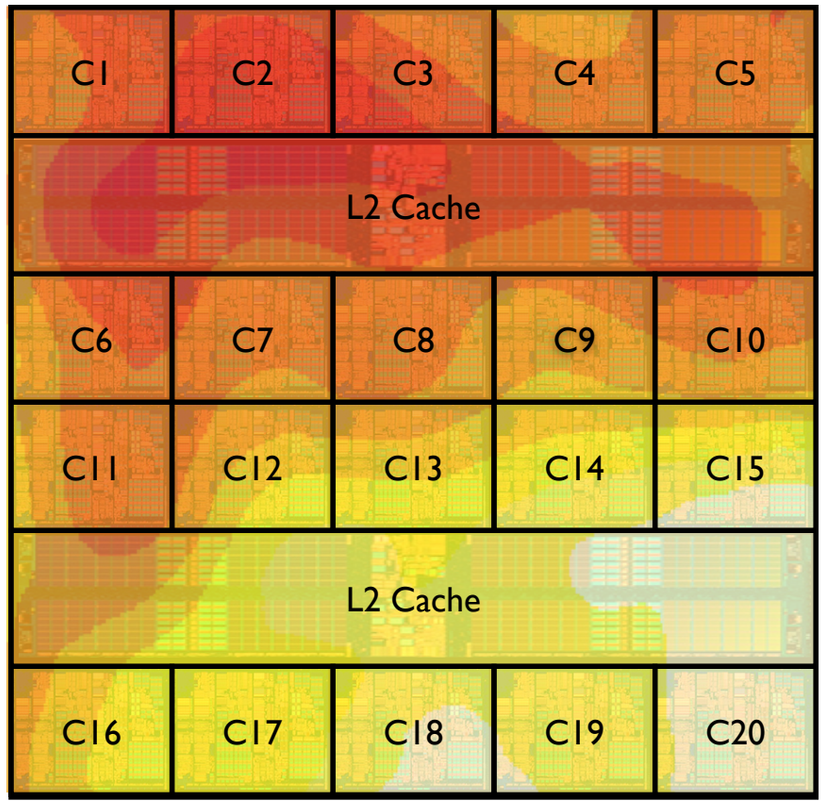
Effects of within-die variation

- CMPs: significant core-to-core variation in frequency and power
- We model a 20-core CMP, 32nm
- On average:

fastest

vs.

slowest



Effects of within-die variation

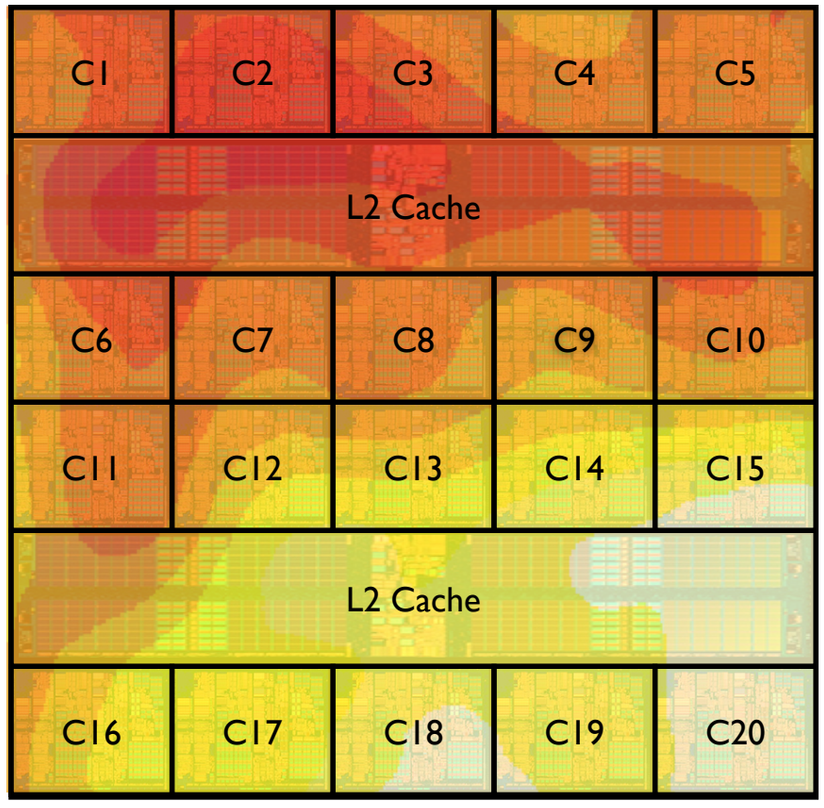
- CMPs: significant core-to-core variation in frequency and power
- We model a 20-core CMP, 32nm
- On average:

Frequency

fastest

vs. 30%

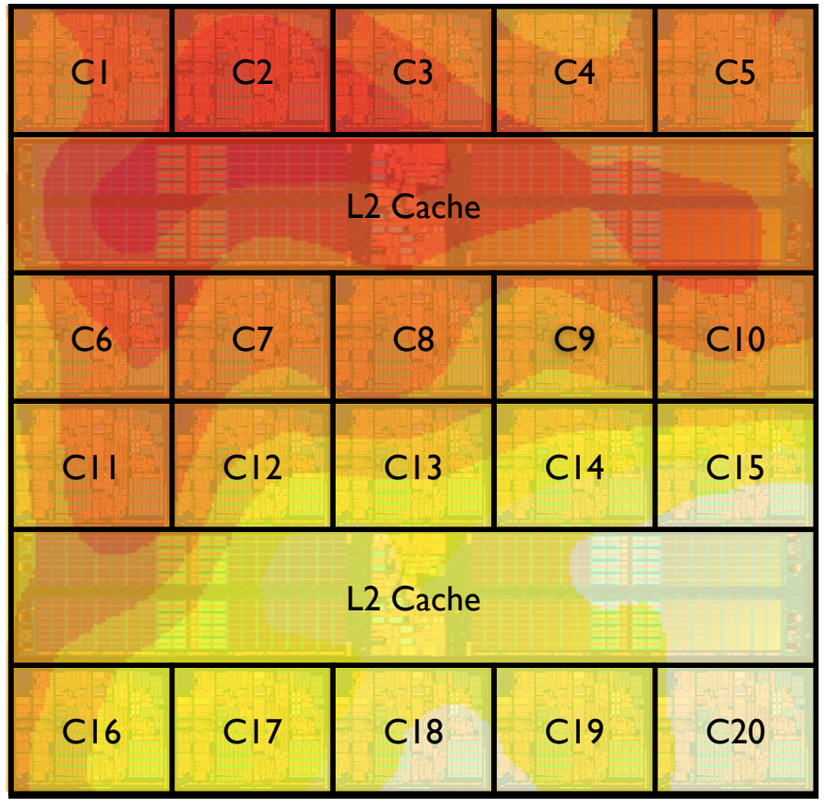
slowest



Effects of within-die variation

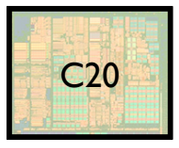
- CMPs: significant core-to-core variation in frequency and power
- We model a 20-core CMP, 32nm
- On average:

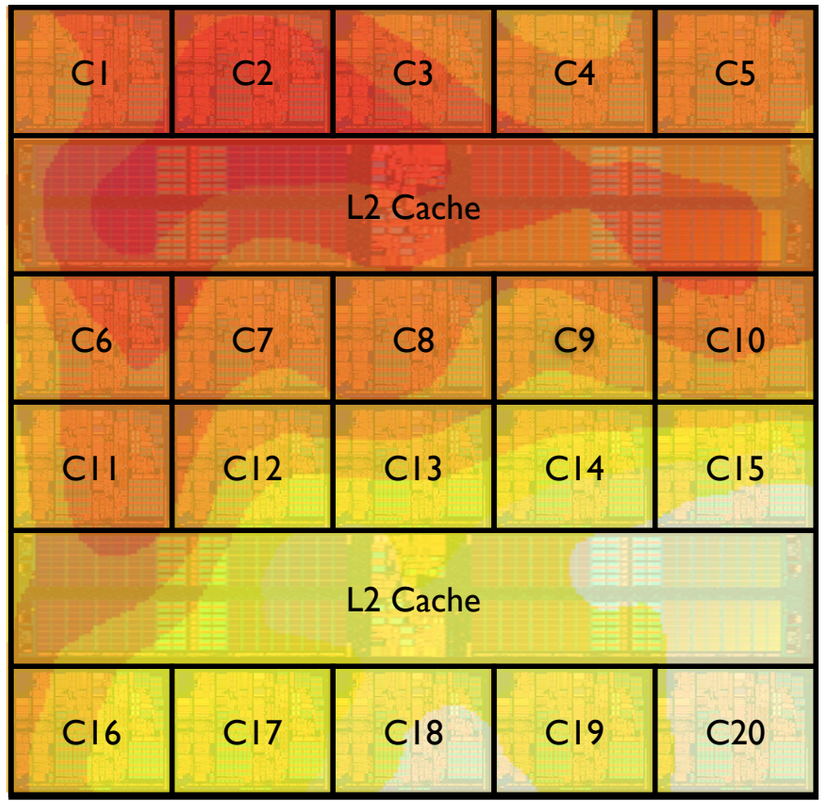
	Frequency	Leakage power
fastest C2		
vs.	30%	2X
slowest C20		



Effects of within-die variation

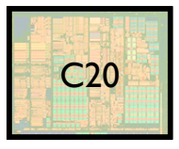
- CMPs: significant core-to-core variation in frequency and power
- We model a 20-core CMP, 32nm
- On average:

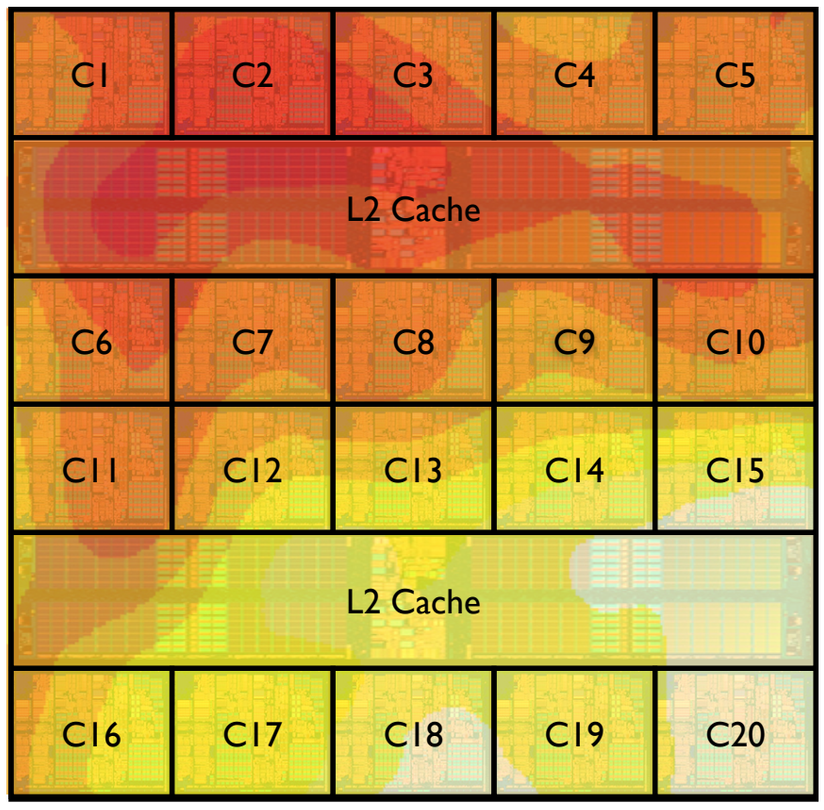
	Frequency	Leakage power	Total power
fastest  vs. slowest 	30%	2X	40%



Effects of within-die variation

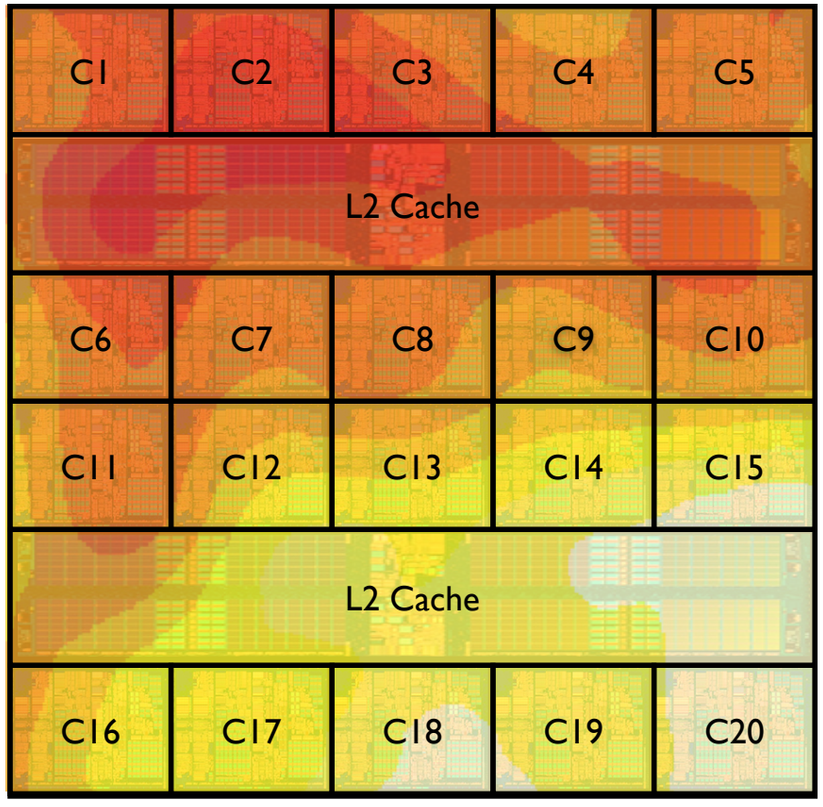
- CMPs: significant core-to-core variation in frequency and power
- We model a 20-core CMP, 32nm
- On average:

	Frequency	Leakage power	Total power
fastest  vs. slowest 	30%	2X	40%



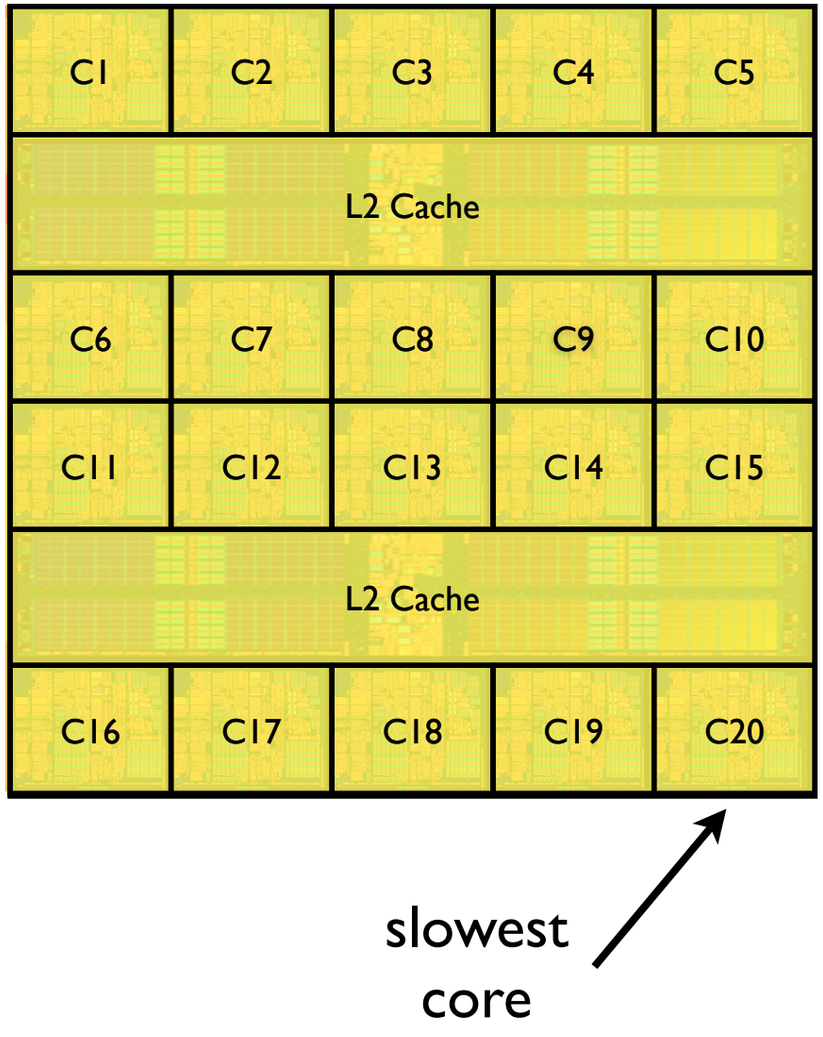
Design-identical cores will have significantly different properties

How can we exploit this variation?



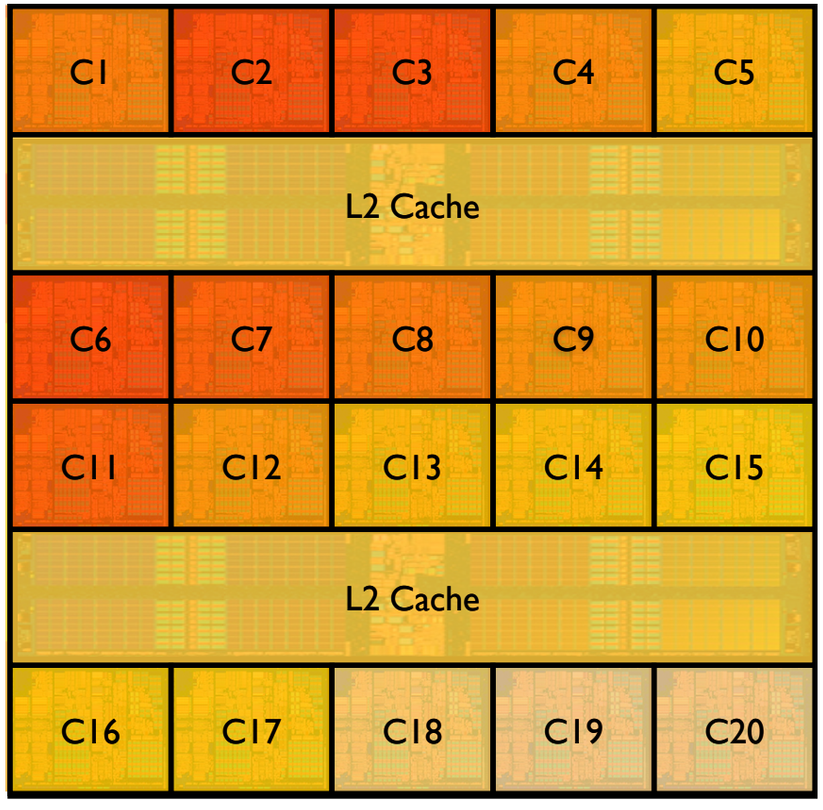
How can we exploit this variation?

- Current CMPs run at the frequency of the slowest core



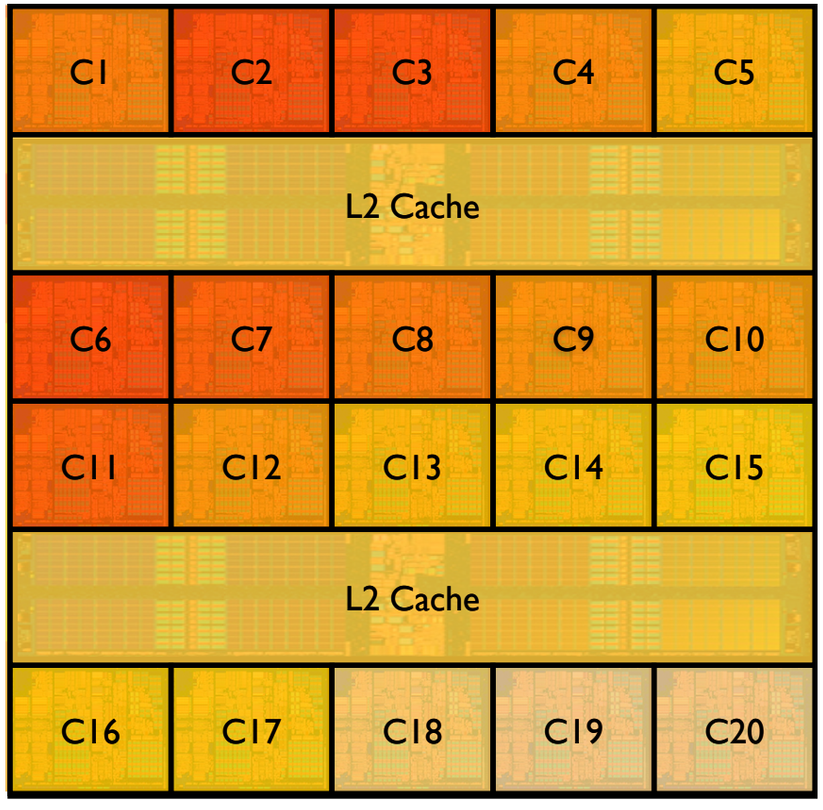
How can we exploit this variation?

- Current CMPs run at the frequency of the slowest core
- We can run each core at the maximum frequency it can achieve



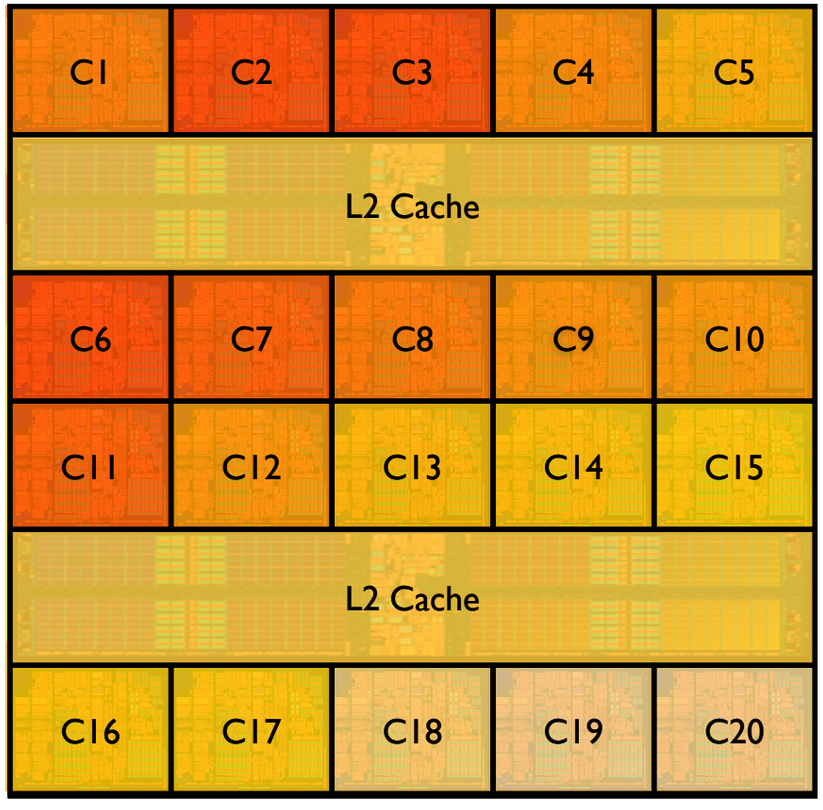
How can we exploit this variation?

- Current CMPs run at the frequency of the slowest core
- We can run each core at the maximum frequency it can achieve
  - 15% average frequency increase



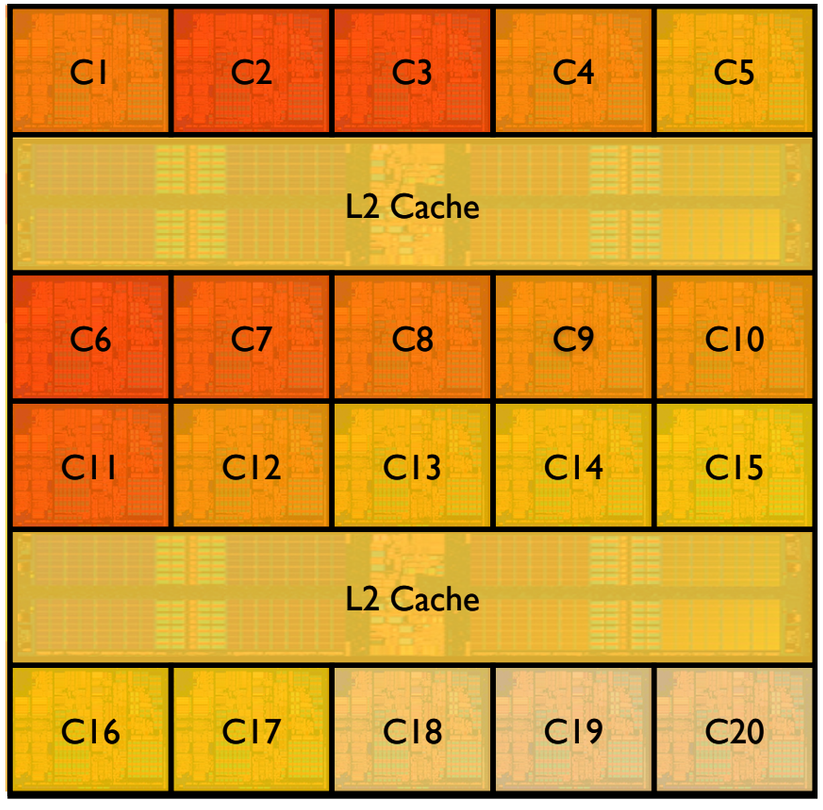
How can we exploit this variation?

- Current CMPs run at the frequency of the slowest core
- We can run each core at the maximum frequency it can achieve
  - 15% average frequency increase
  - Support present in AMD's 4-core Opteron



How can we exploit this variation?

- Current CMPs run at the frequency of the slowest core
- We can run each core at the maximum frequency it can achieve
  - 15% average frequency increase
  - Support present in AMD's 4-core Opteron
- Heterogeneous system



Contributions

Contributions

- Expose variation in core frequency and power to the OS

Contributions

- Expose variation in core frequency and power to the OS
- Variation-aware application scheduling algorithms

Contributions

- Expose variation in core frequency and power to the OS
- Variation-aware application scheduling algorithms
- Variation-aware global power management subsystem

Contributions

- Expose variation in core frequency and power to the OS
- Variation-aware application scheduling algorithms
- Variation-aware global power management subsystem
  - On-line optimization algorithm that maximizes system performance at a power budget

Contributions

- Expose variation in core frequency and power to the OS
- Variation-aware application scheduling algorithms
- Variation-aware global power management subsystem
  - On-line optimization algorithm that maximizes system performance at a power budget
  - 12-17% CMP throughput improvement at the same power

Outline

Outline

- Variation-aware scheduling

Outline

- Variation-aware scheduling
- Variation-aware power management
  - Defining the optimization problem
  - Implementation

Outline

- Variation-aware scheduling
- Variation-aware power management
  - Defining the optimization problem
  - Implementation
- Evaluation

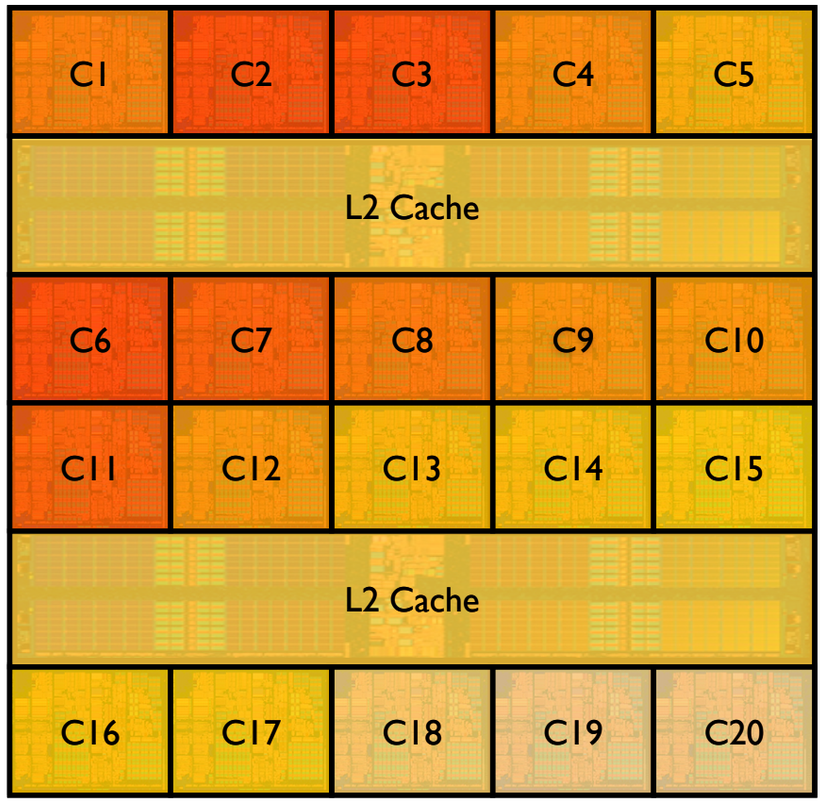
Outline

- Variation-aware scheduling
- Variation-aware power management
  - Defining the optimization problem
  - Implementation
- Evaluation
- Conclusions

Outline

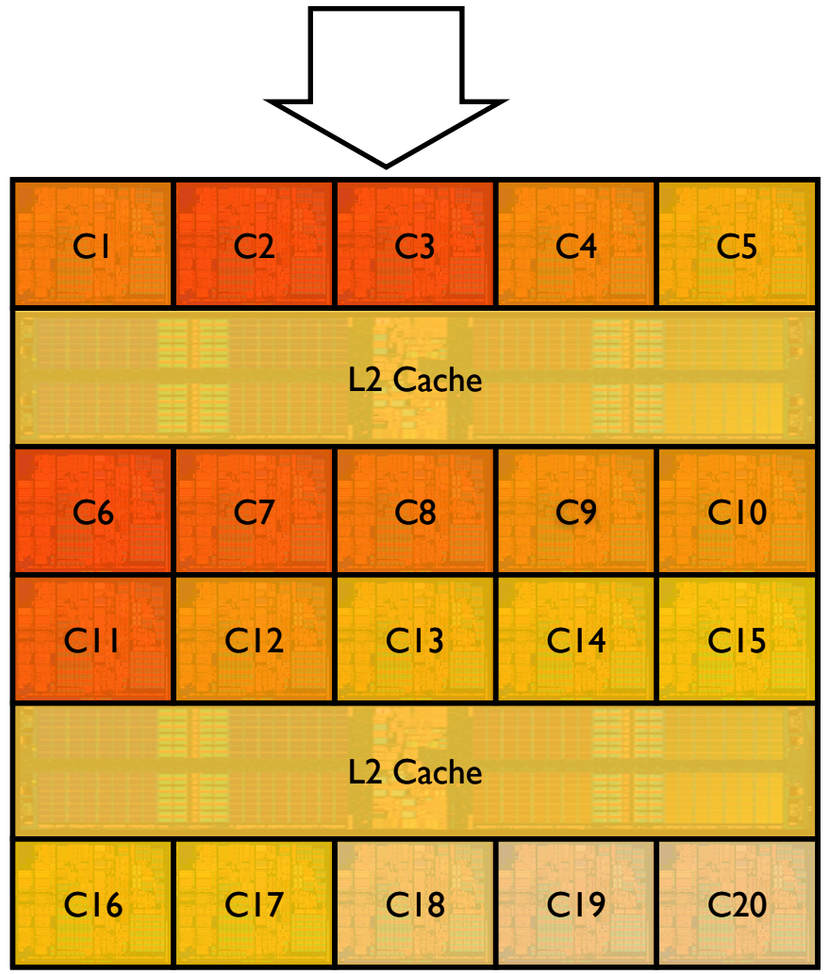
- Variation-aware scheduling
- Variation-aware power management
  - Defining the optimization problem
  - Implementation
- Evaluation
- Conclusions

Variation-aware scheduling

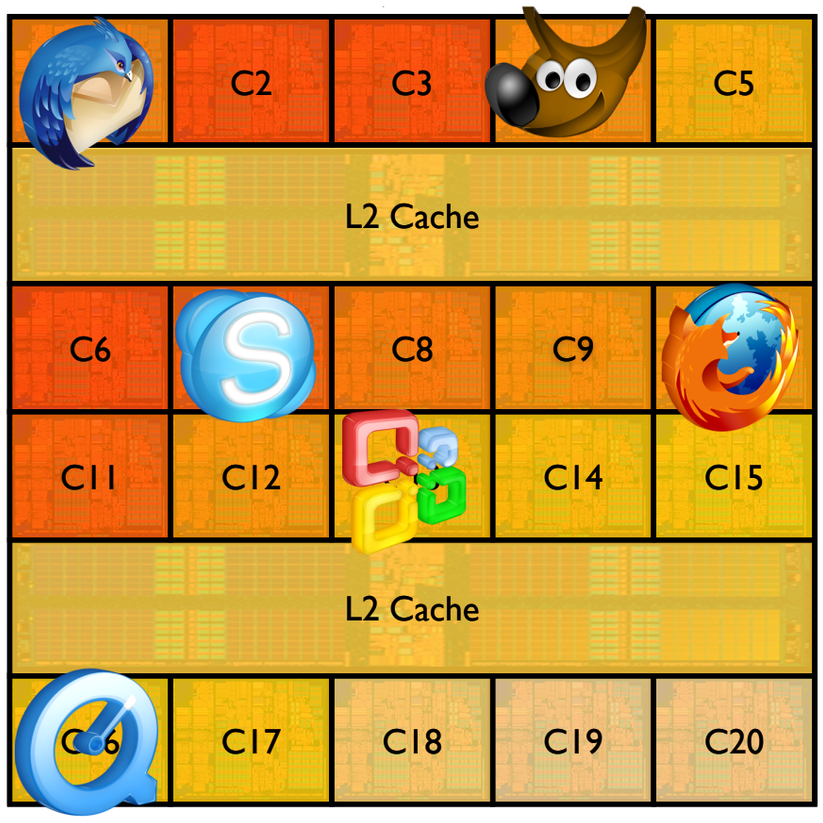


Variation-aware scheduling

Applications

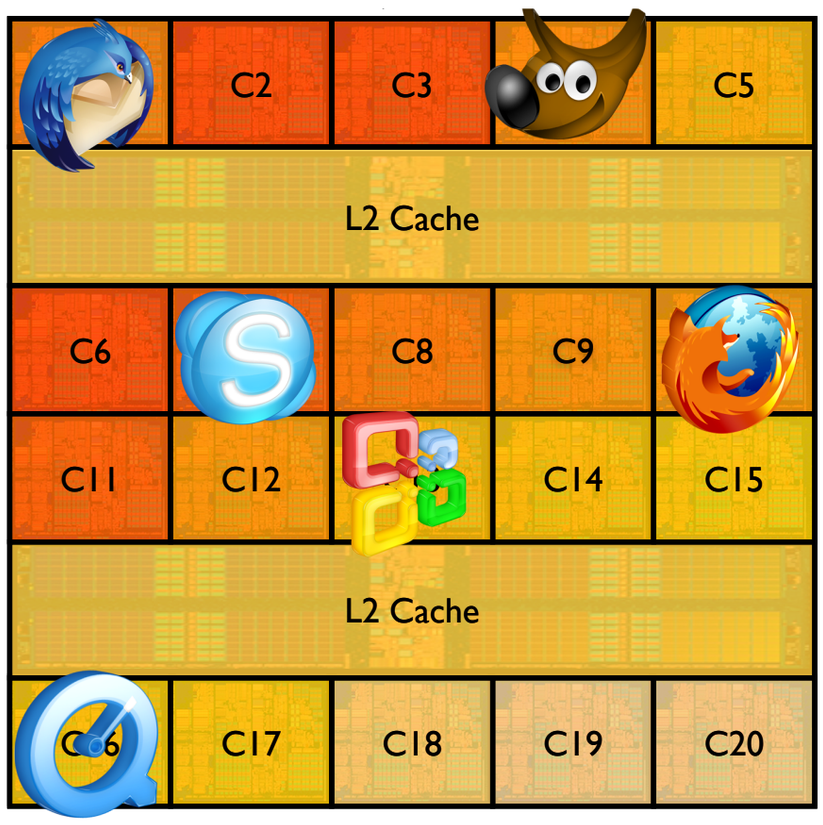


Variation-aware scheduling



Variation-aware scheduling

Additional information to guide scheduling decisions:



Variation-aware scheduling

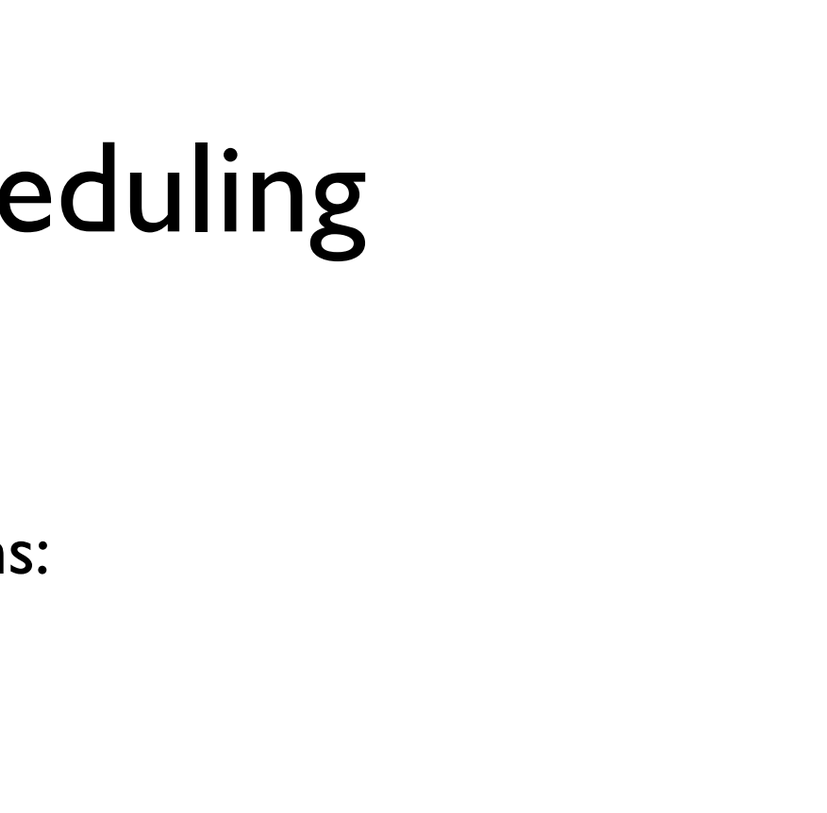
Additional information to guide scheduling decisions:

- Per core frequency and static power

Variation-aware scheduling

Additional information to guide scheduling decisions:

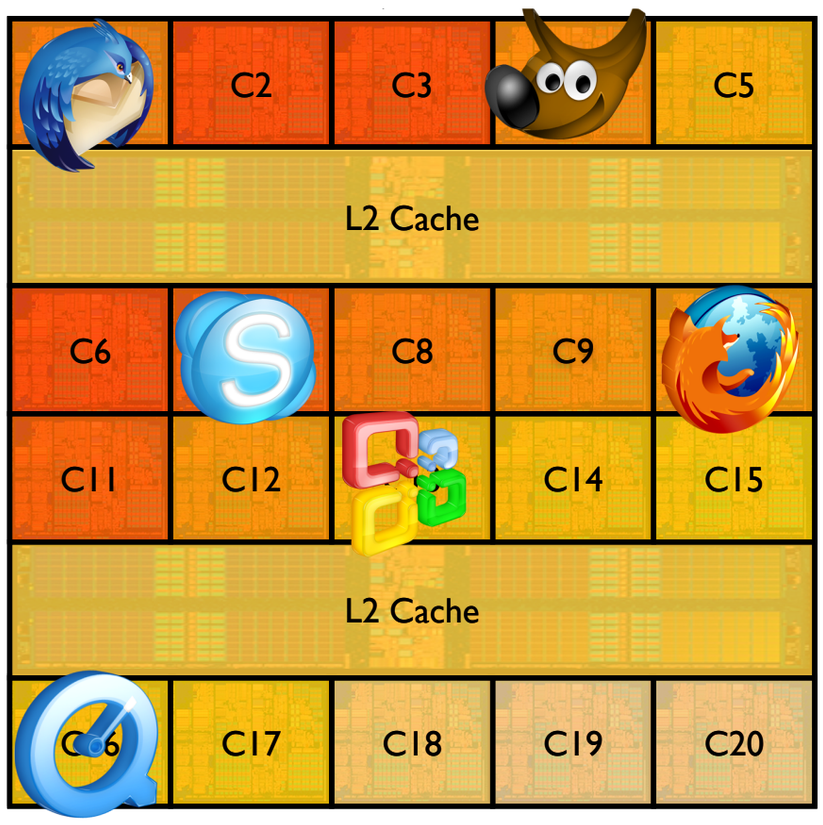
- Per core frequency and static power
- Application behavior



Variation-aware scheduling

Additional information to guide scheduling decisions:

- Per core frequency and static power
- Application behavior
- Dynamic power consumption



Variation-aware scheduling

Additional information to guide scheduling decisions:

- Per core frequency and static power
- Application behavior
  - Dynamic power consumption
  - Compute intensity (IPC)

Variation-aware scheduling

Additional information to guide scheduling decisions:

- Per core frequency and static power
- Application behavior
  - Dynamic power consumption
  - Compute intensity (IPC)
- Multiple possible goals:

Variation-aware scheduling

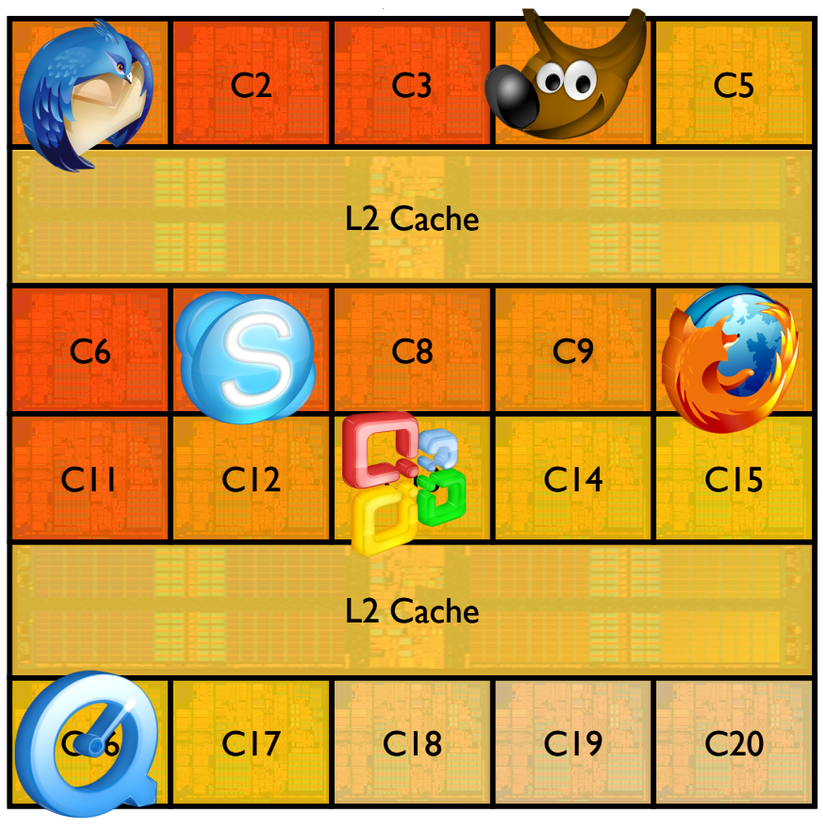
Additional information to guide scheduling decisions:

- Per core frequency and static power
- Application behavior
  - Dynamic power consumption
  - Compute intensity (IPC)
- Multiple possible goals:
  - Reduce power

Variation-aware scheduling

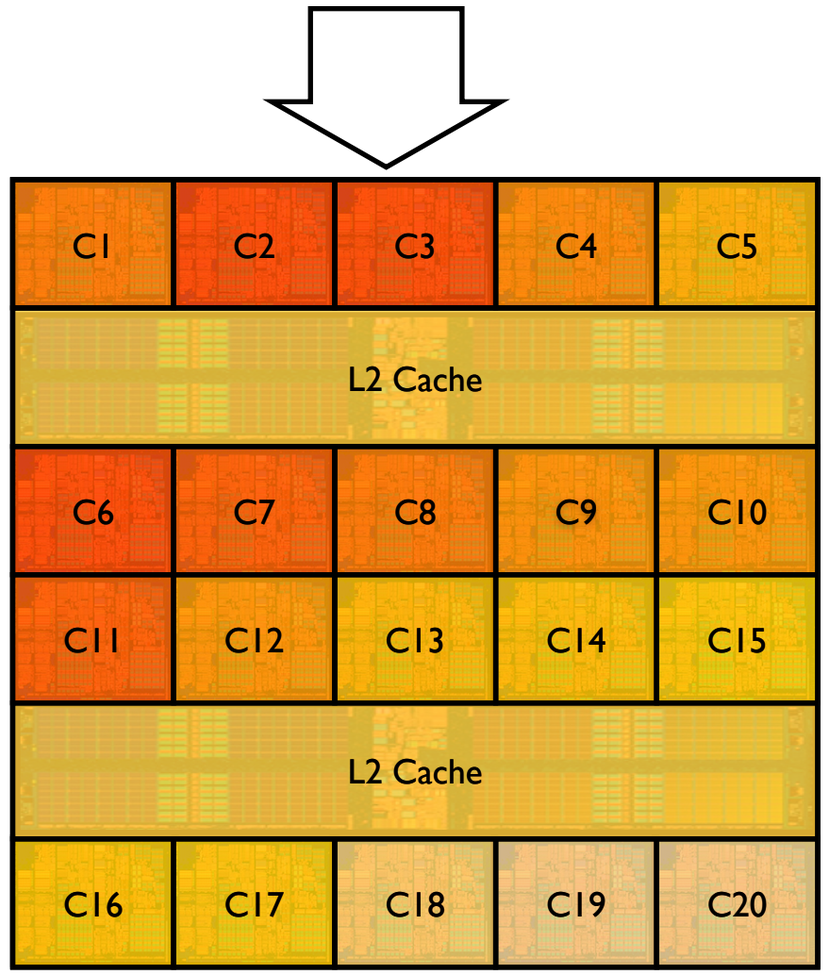
Additional information to guide scheduling decisions:

- Per core frequency and static power
- Application behavior
  - Dynamic power consumption
  - Compute intensity (IPC)
- Multiple possible goals:
  - Reduce power
  - Improve performance



Variation-aware scheduling

When the goal is to reduce power consumption:

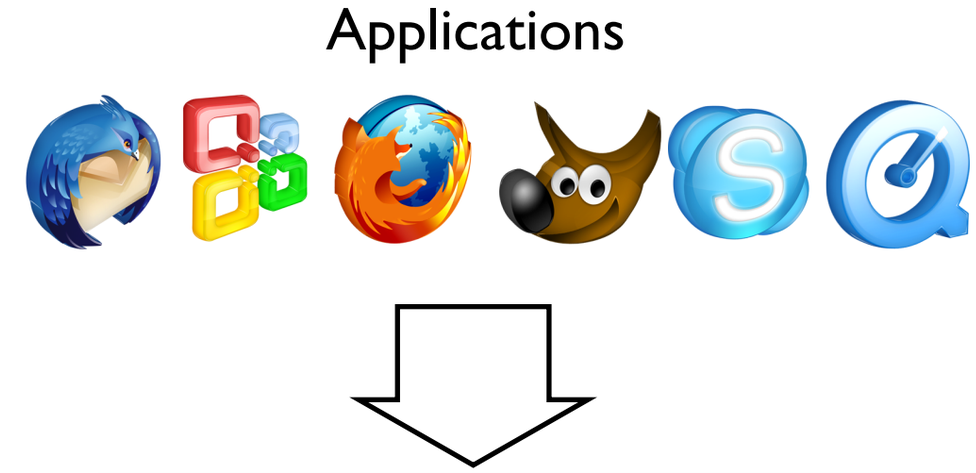


Variation-aware scheduling

When the goal is to reduce power consumption:

- ***VarP***

 Assign applications to low static power cores first



Variation-aware scheduling

When the goal is to reduce power consumption:

- ***VarP***

  Assign applications to low static power cores first

Variation-aware scheduling

When the goal is to reduce power consumption:

- ***VarP***

  Assign applications to low static power cores first

- ***VarP&AppP***

  Assign applications with high dynamic power to low static power cores

Variation-aware scheduling

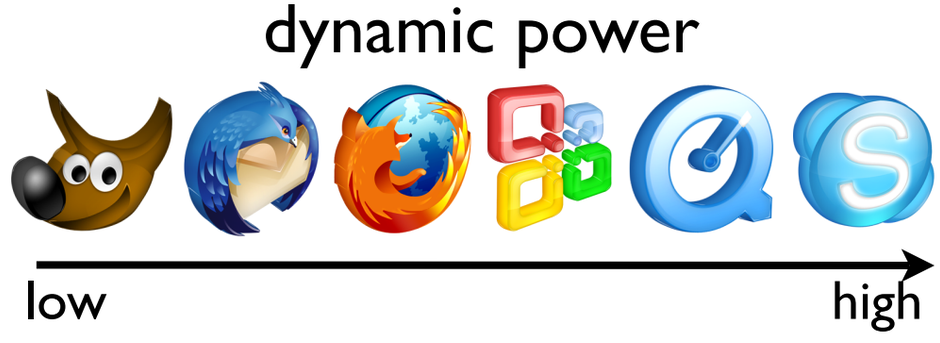
When the goal is to reduce power consumption:

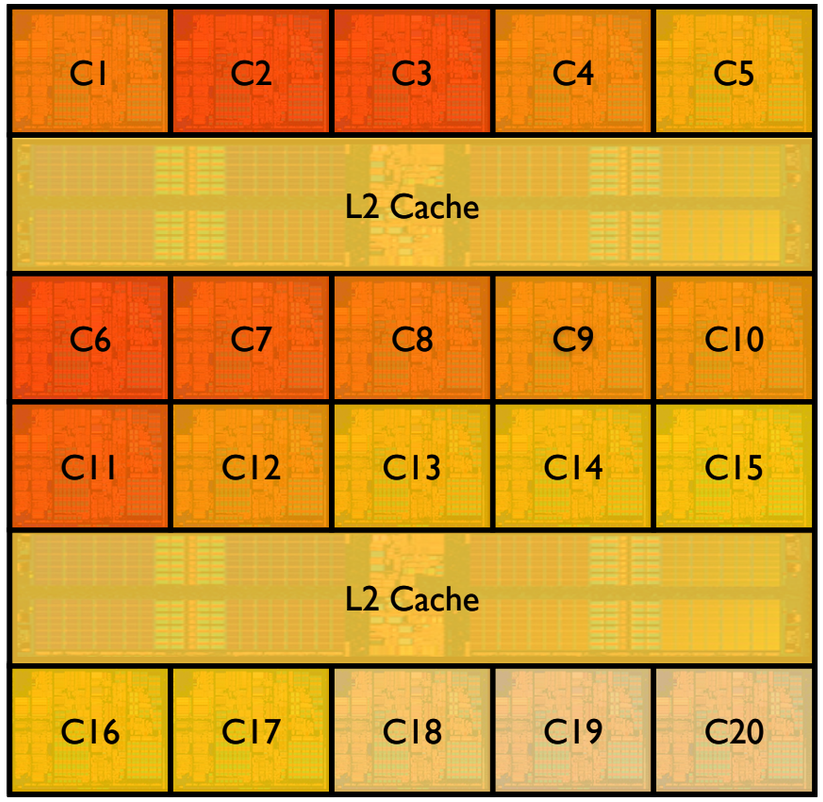
- ***VarP***

  Assign applications to low static power cores first

- ***VarP&AppP***

  Assign applications with high dynamic power to low static power cores





Variation-aware scheduling

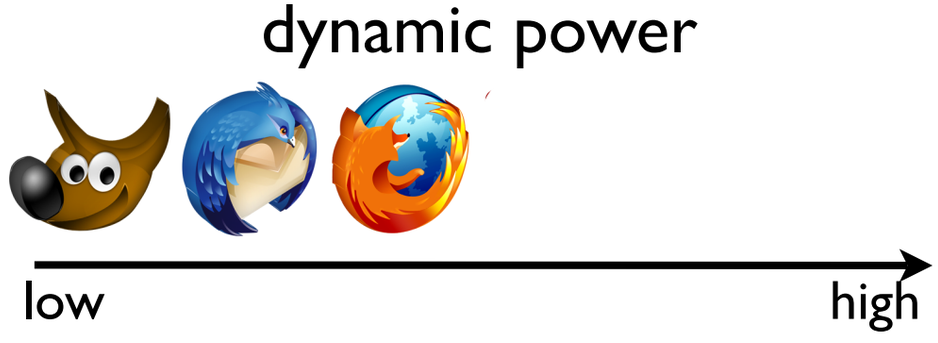
When the goal is to reduce power consumption:

- ***VarP***

  Assign applications to low static power cores first

- ***VarP&AppP***

  Assign applications with high dynamic power to low static power cores



Variation-aware scheduling

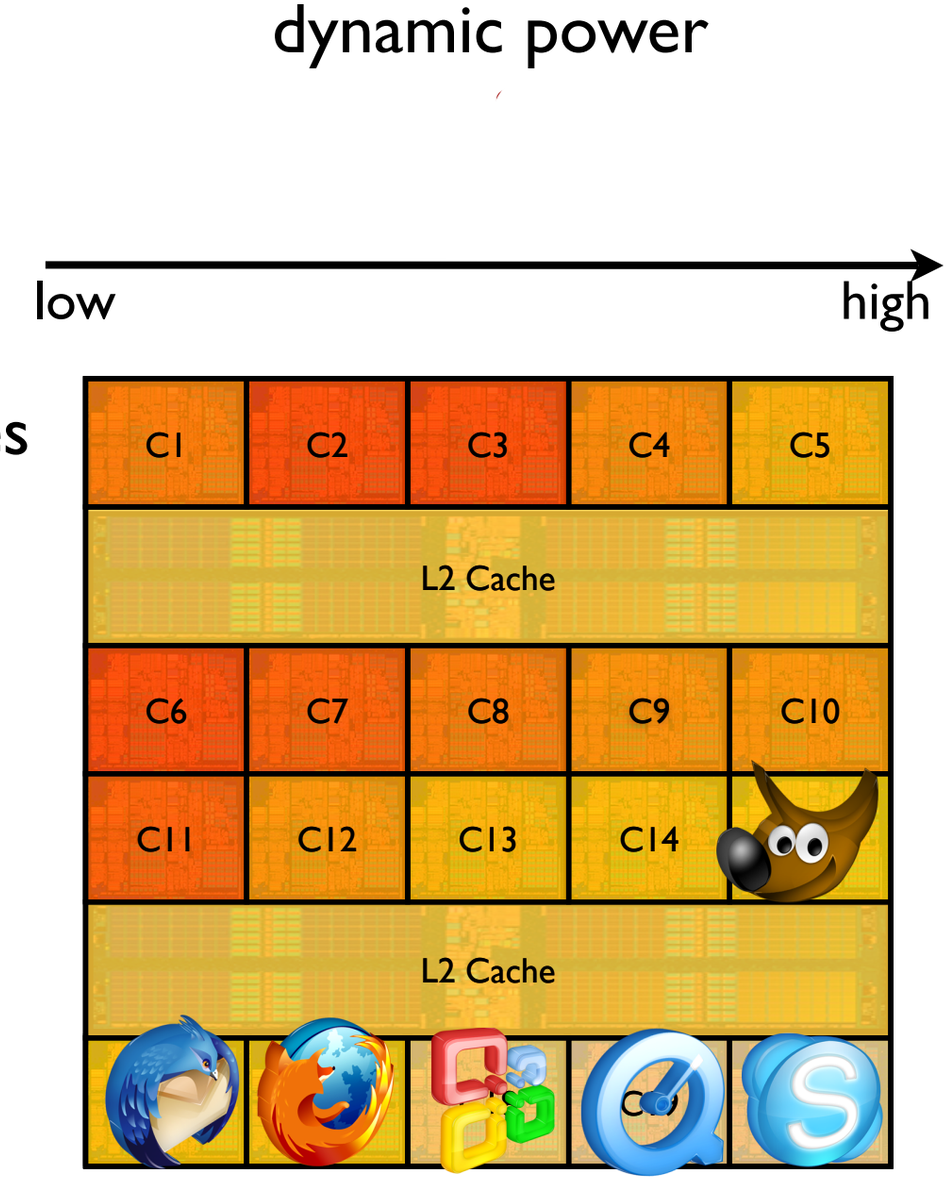
When the goal is to reduce power consumption:

- ***VarP***

  Assign applications to low static power cores first

- ***VarP&AppP***

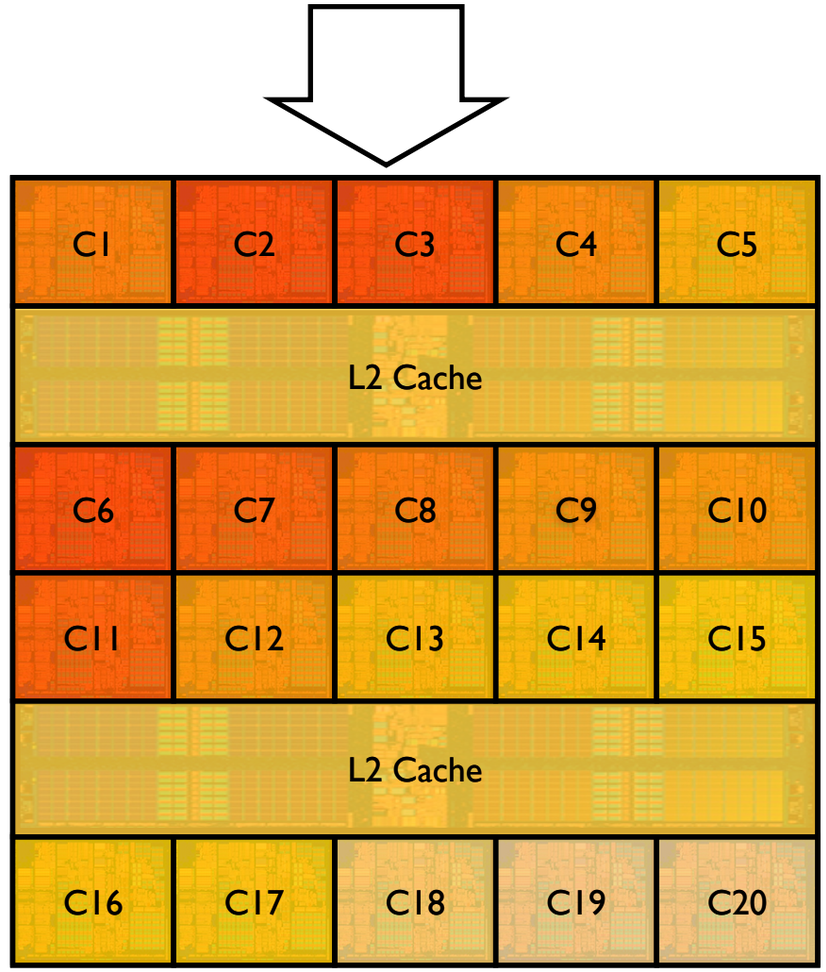
  Assign applications with high dynamic power to low static power cores



Variation-aware scheduling

When the goal is to improve performance:

Applications

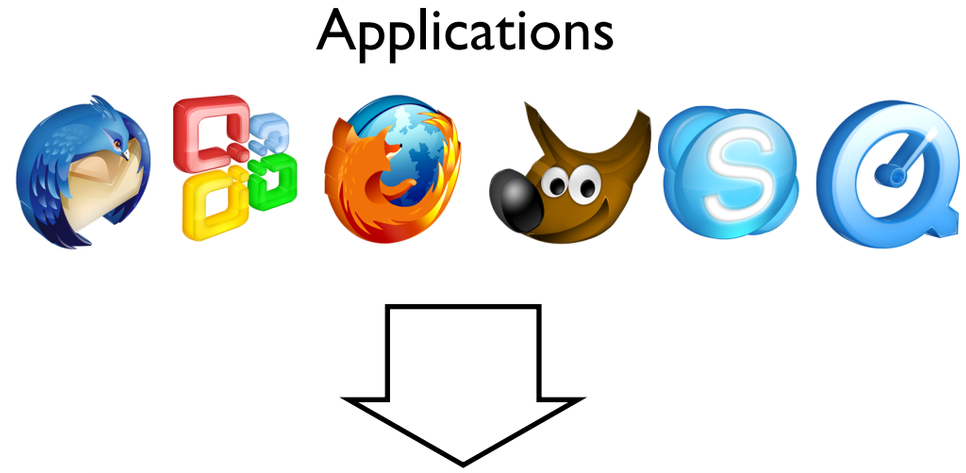


Variation-aware scheduling

When the goal is to improve performance:

- ***VarF***

 Assign applications to high frequeny cores first



Variation-aware scheduling

When the goal is to improve performance:

- ***VarF***

  Assign applications to high frequeny cores first

Variation-aware scheduling

When the goal is to improve performance:

- ***VarF***

  Assign applications to high frequeny cores first

- ***VarF&AppIPC***

  Assign high IPC applications to high frequency cores

Variation-aware scheduling

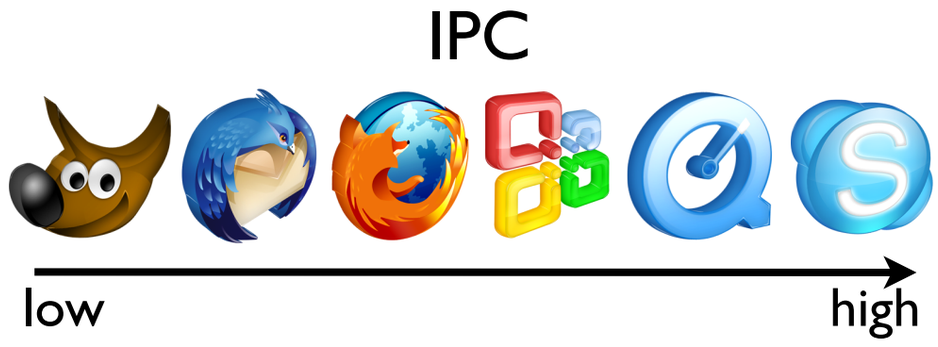
When the goal is to improve performance:

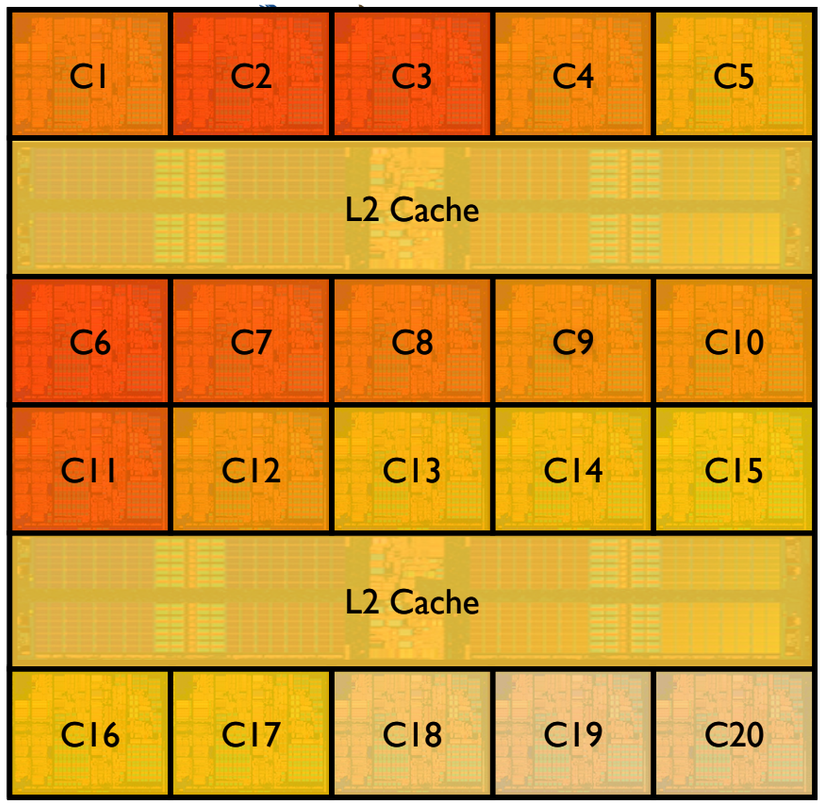
- ***VarF***

  Assign applications to high frequeny cores first

- ***VarF&AppIPC***

  Assign high IPC applications to high frequency cores





Variation-aware scheduling

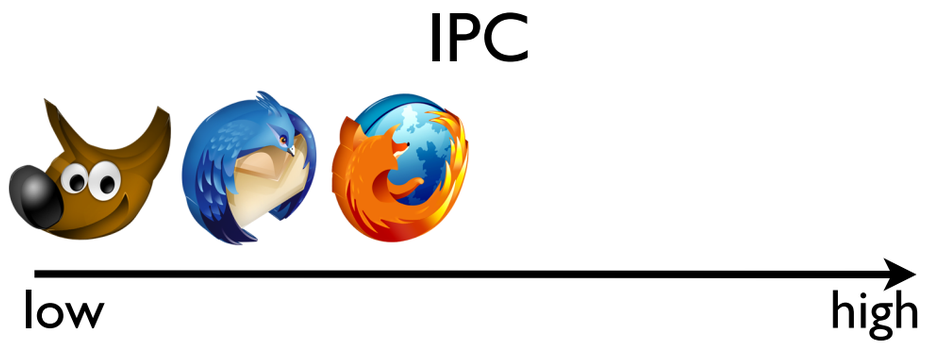
When the goal is to improve performance:

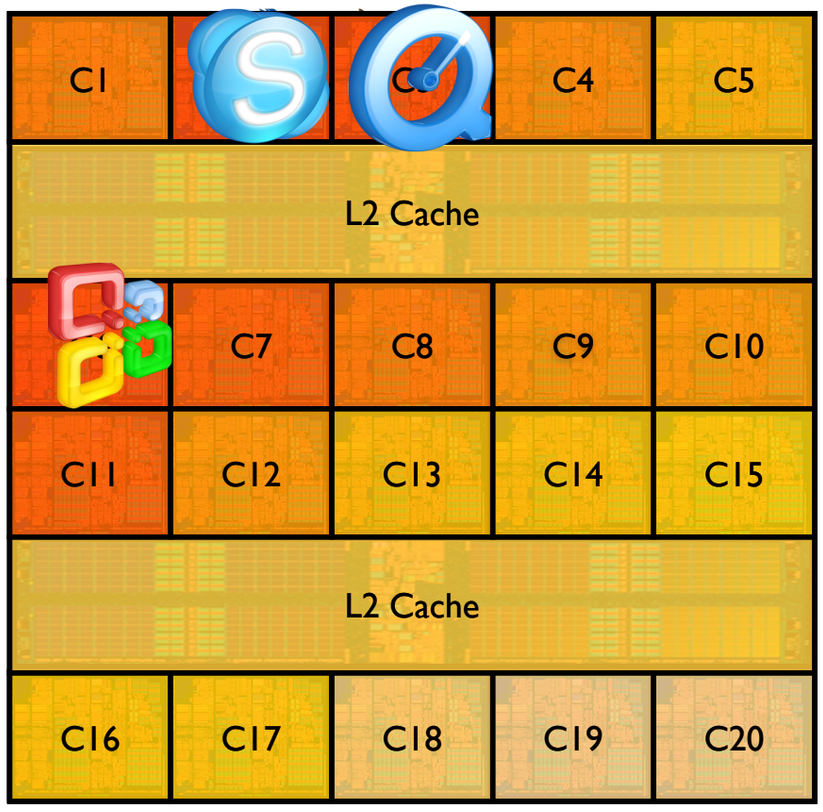
- ***VarF***

  Assign applications to high frequeny cores first

- ***VarF&AppIPC***

  Assign high IPC applications to high frequency cores





Variation-aware scheduling

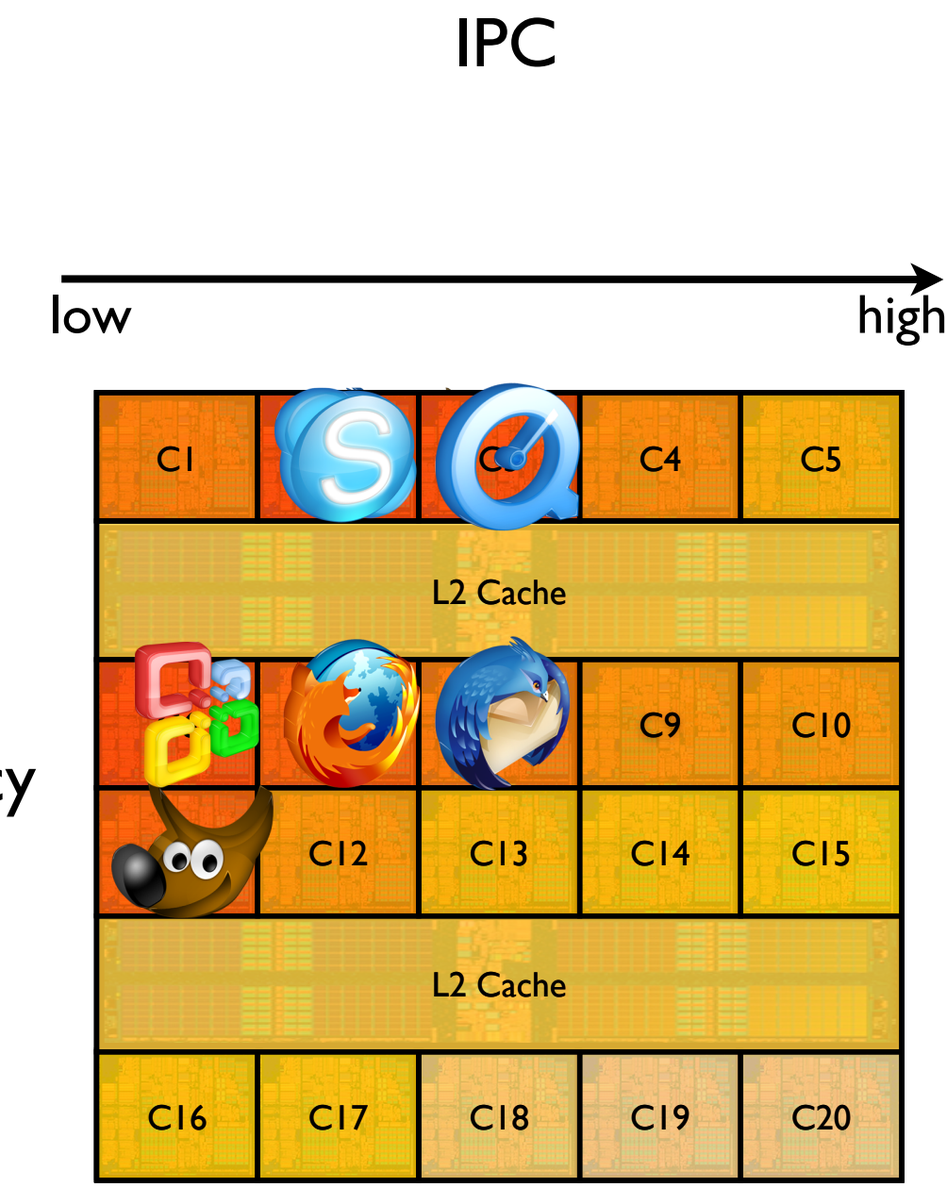
When the goal is to improve performance:

- ***VarF***

  Assign applications to high frequeny cores first

- ***VarF&AppIPC***

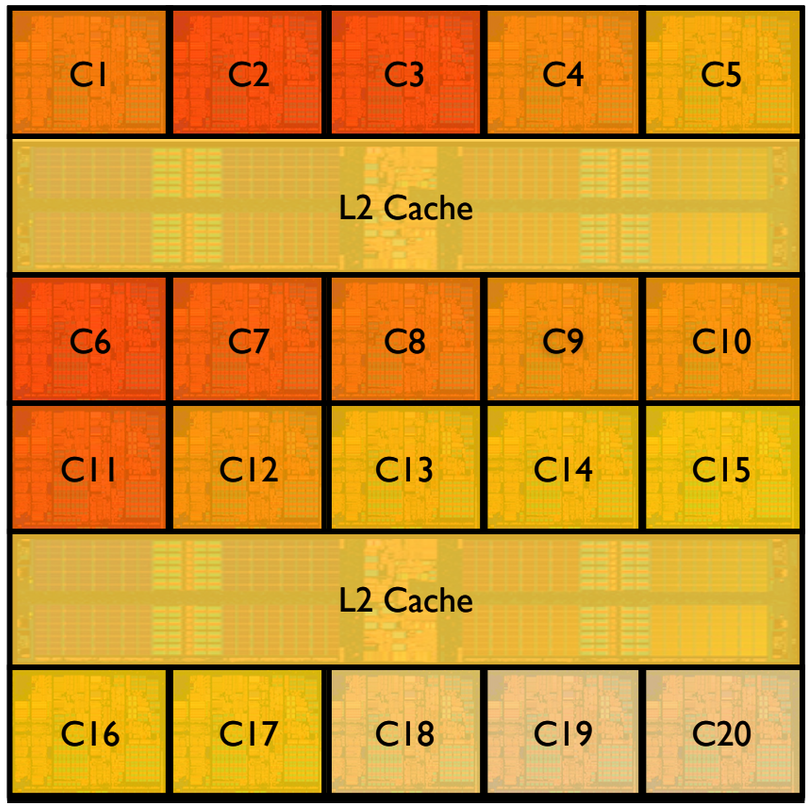
  Assign high IPC applications to high frequency cores



Outline

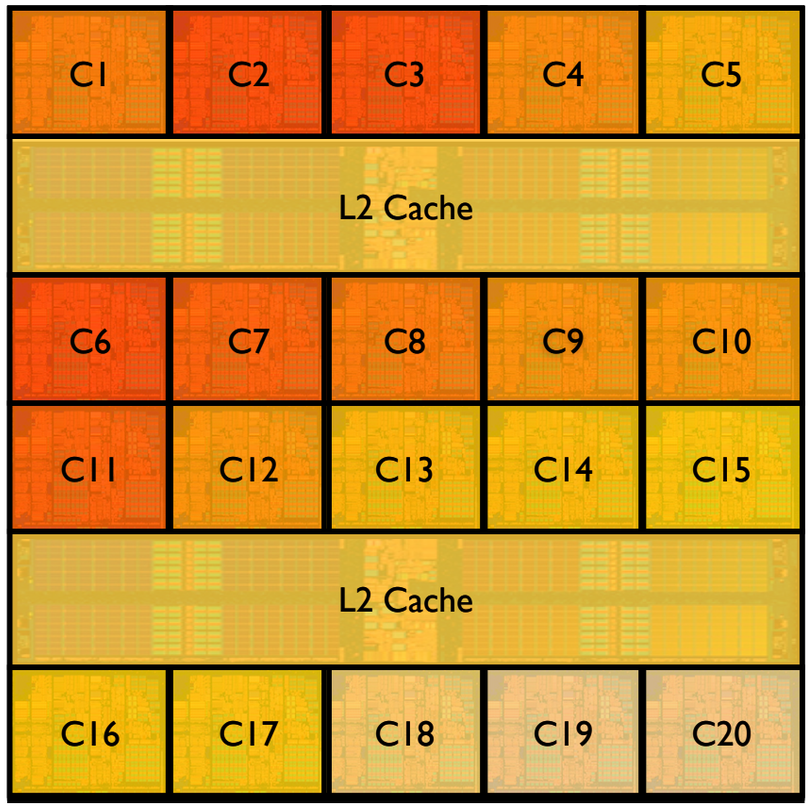
- Variation-aware scheduling
- Variation-aware power management
  - Defining the optimization problem
  - Implementation
- Evaluation
- Conclusions

Variation-aware global power management



Variation-aware global power management

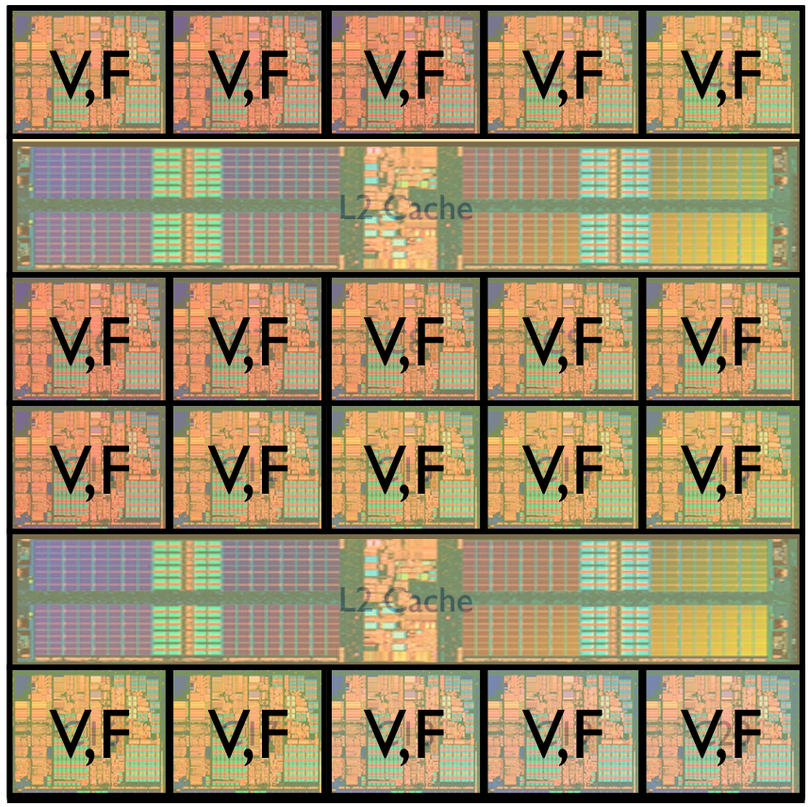
CMP power management



Variation-aware global power management

CMP power management

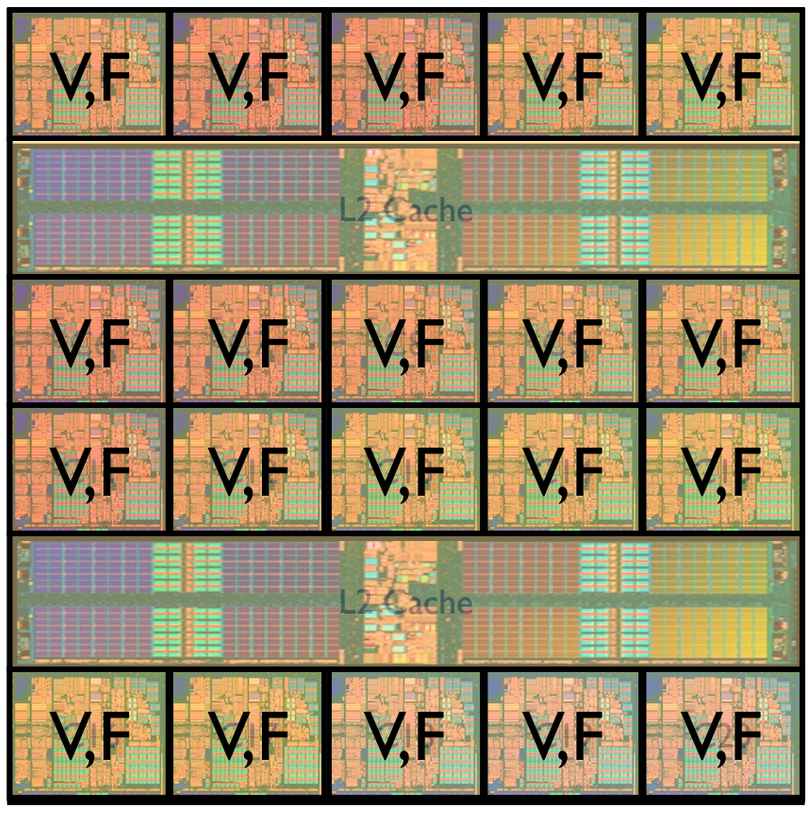
- Per core dynamic voltage and frequency scaling (DVFS)



Variation-aware global power management

CMP power management

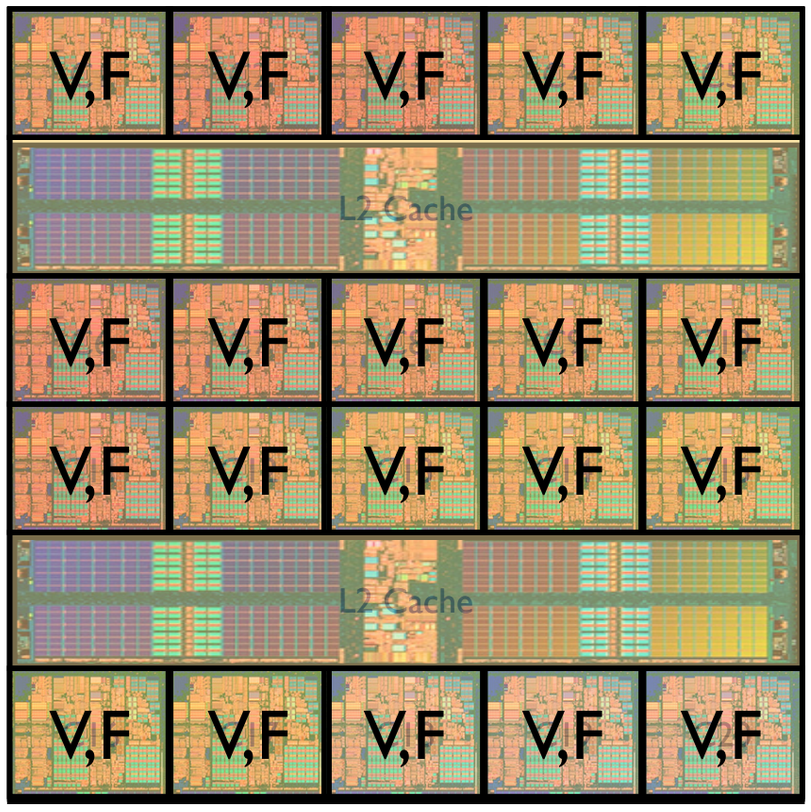
- Per core dynamic voltage and frequency scaling (DVFS)
- Challenge: find best (V,F) for each core



Variation-aware global power management

CMP power management

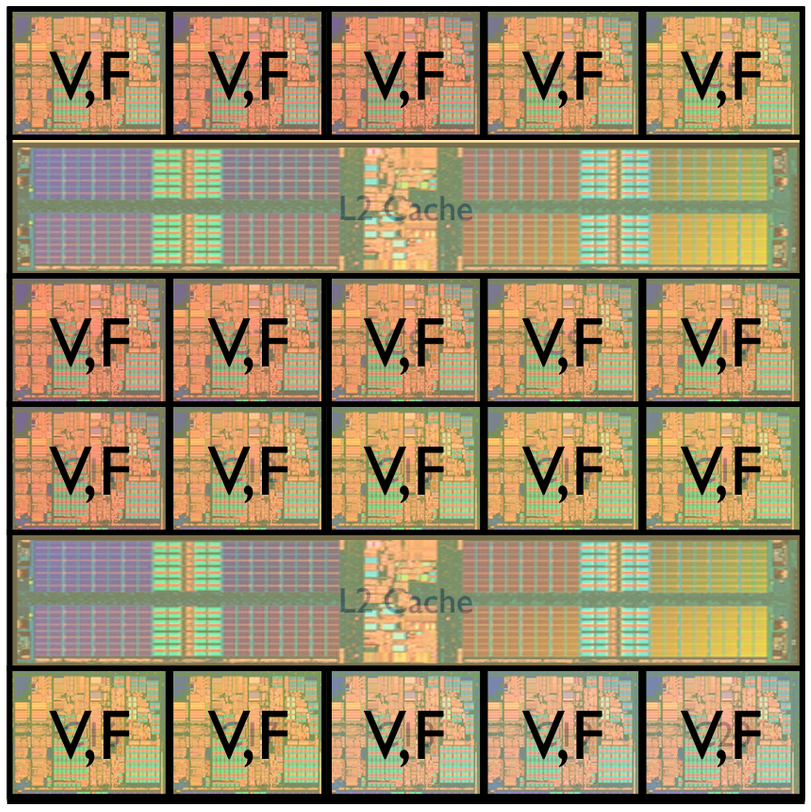
- Per core dynamic voltage and frequency scaling (DVFS)
- Challenge: find best (V,F) for each core
  - Core-level decisions less effective in large CMPs



Variation-aware global power management

CMP power management

- Per core dynamic voltage and frequency scaling (DVFS)
- Challenge: find best (V,F) for each core
  - Core-level decisions less effective in large CMPs
  - Global (CMP-wide) power management solution is needed

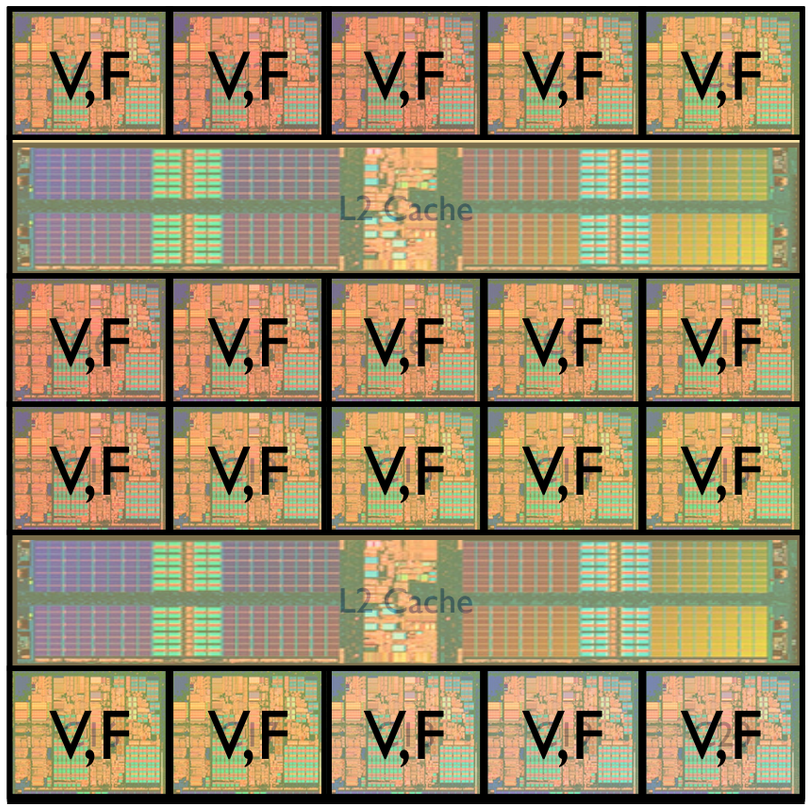


Variation-aware global power management

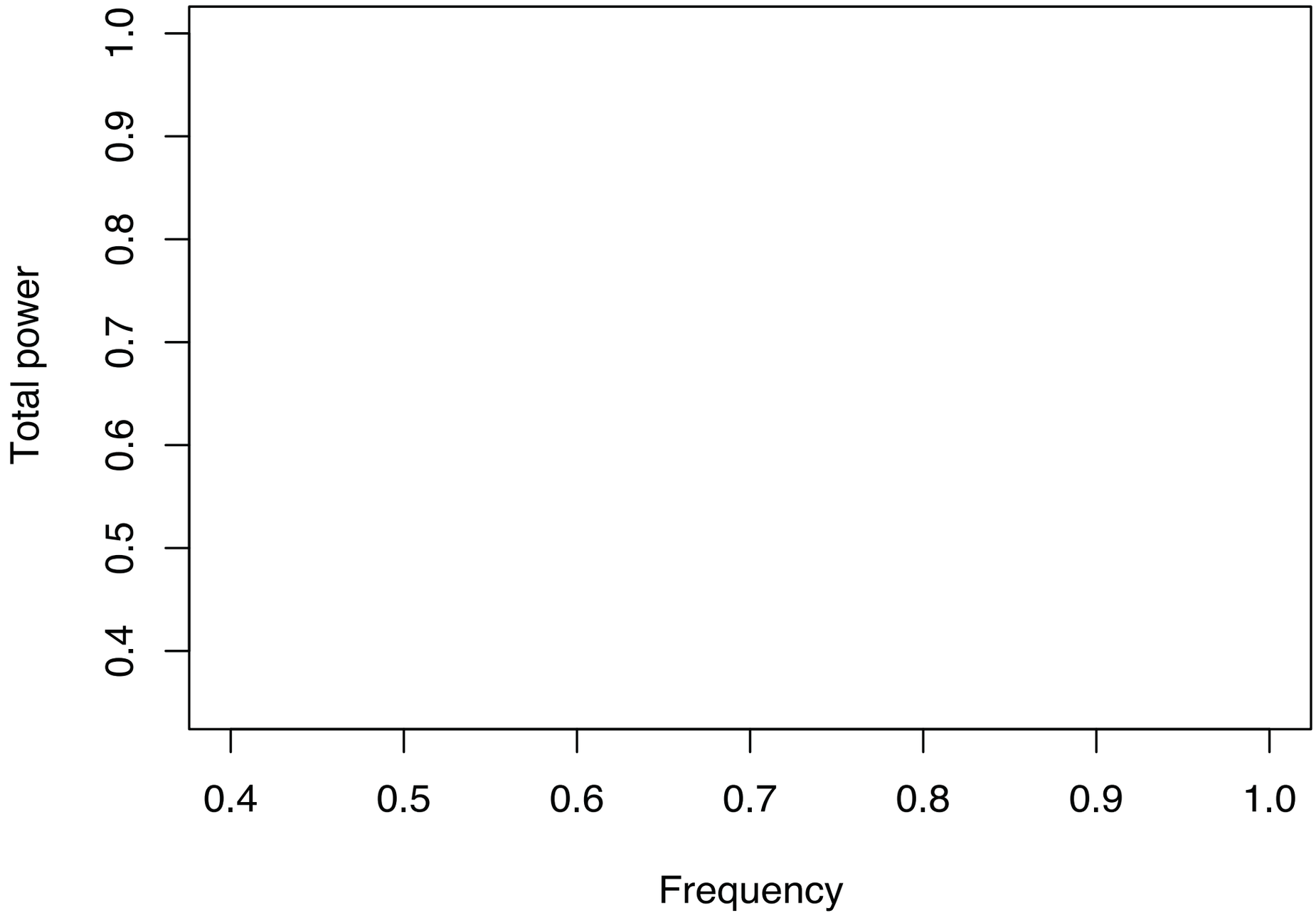
CMP power management

- Per core dynamic voltage and frequency scaling (DVFS)
- Challenge: find best (V,F) for each core
  - Core-level decisions less effective in large CMPs
  - Global (CMP-wide) power management solution is needed

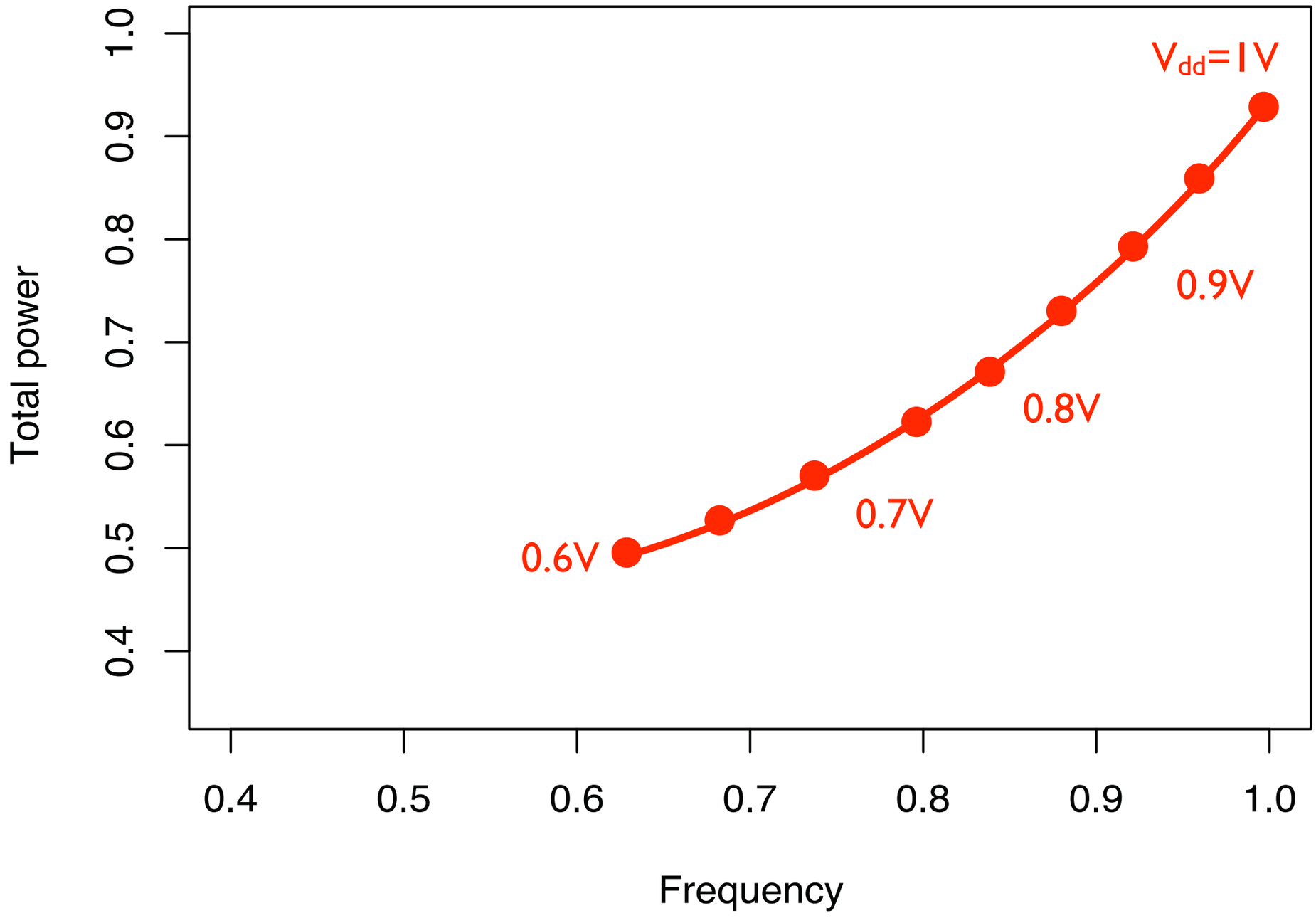
Variation makes the problem more difficult



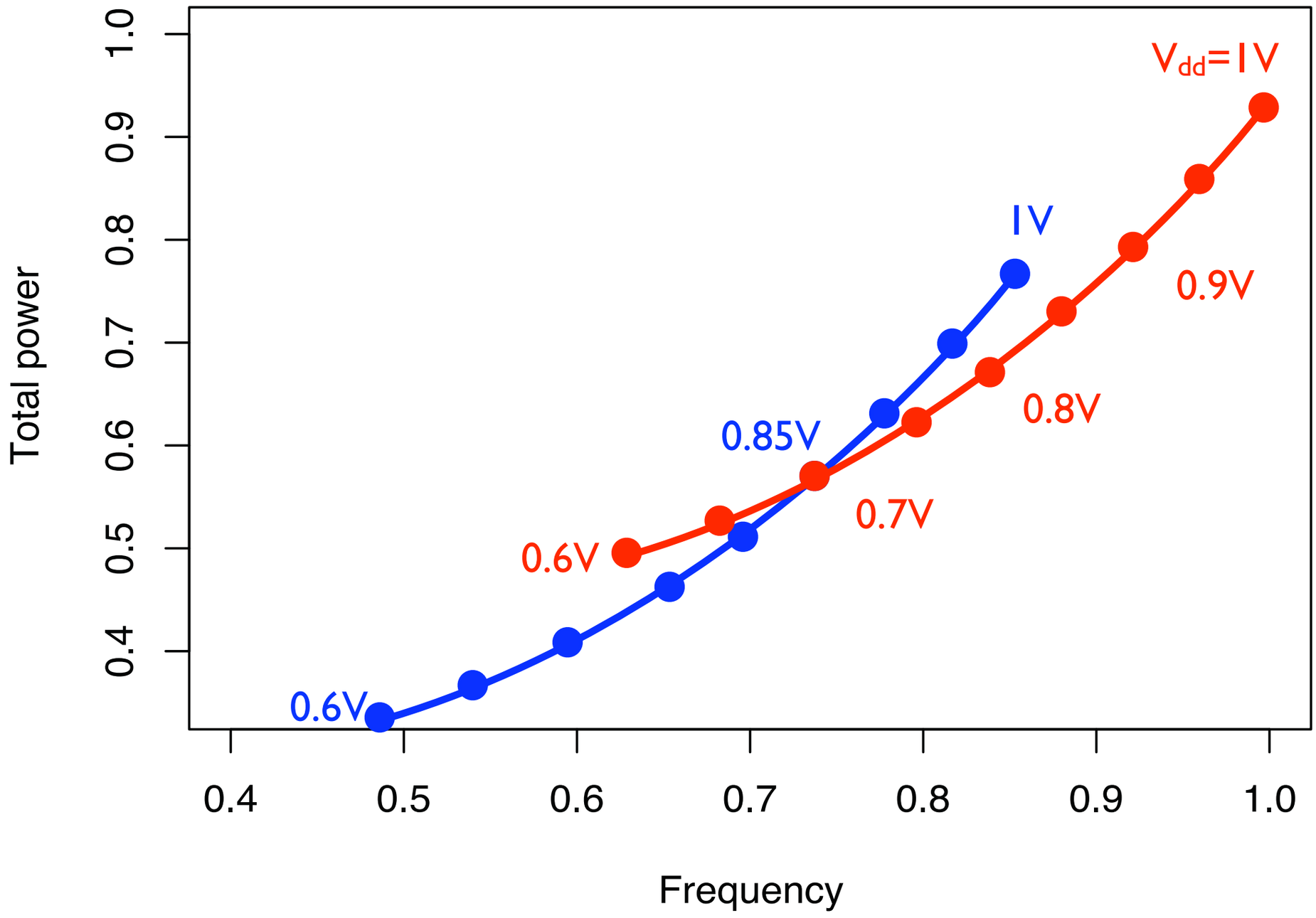
DVFS under variation



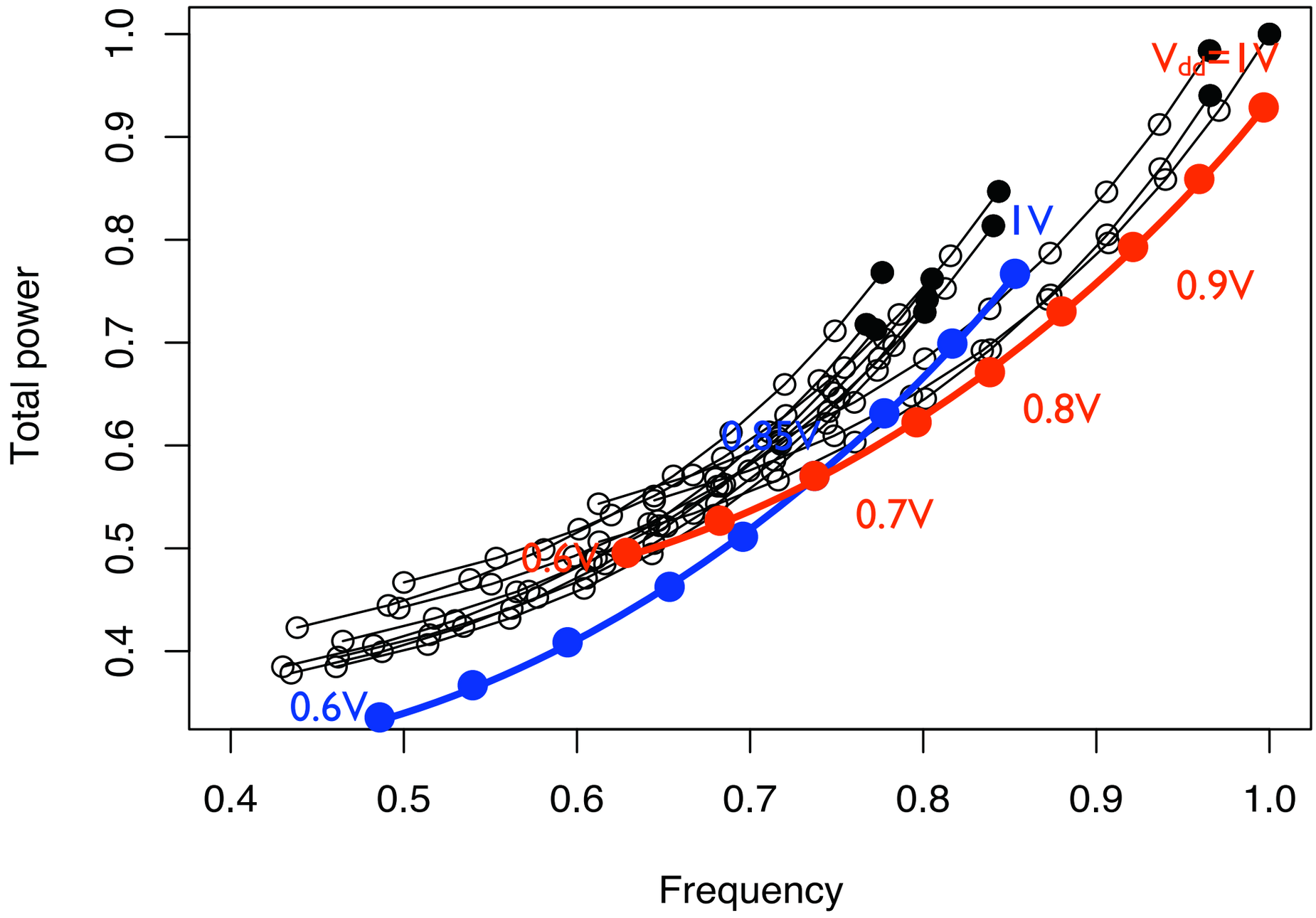
DVFS under variation



DVFS under variation



DVFS under variation



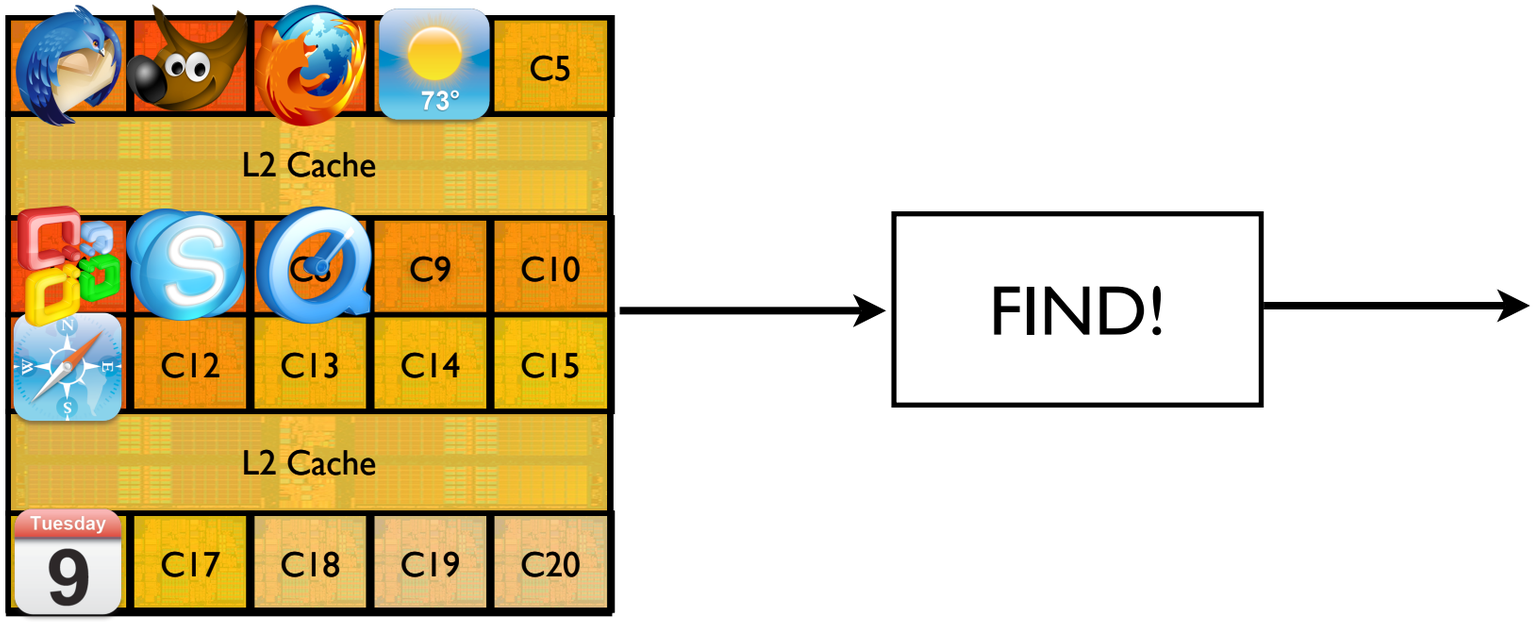
Optimization problem

Optimization problem

Given a mapping of threads to cores (variation-aware):

Optimization problem

Given a mapping of threads to cores (variation-aware):

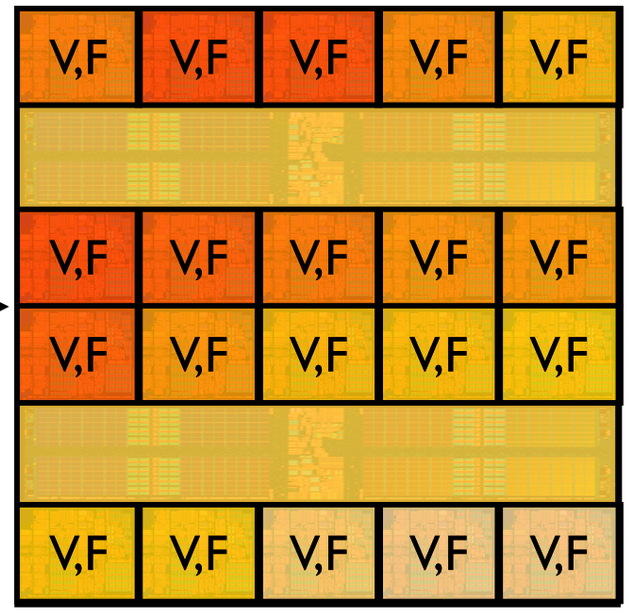


Optimization problem

Given a mapping of threads to cores (variation-aware):

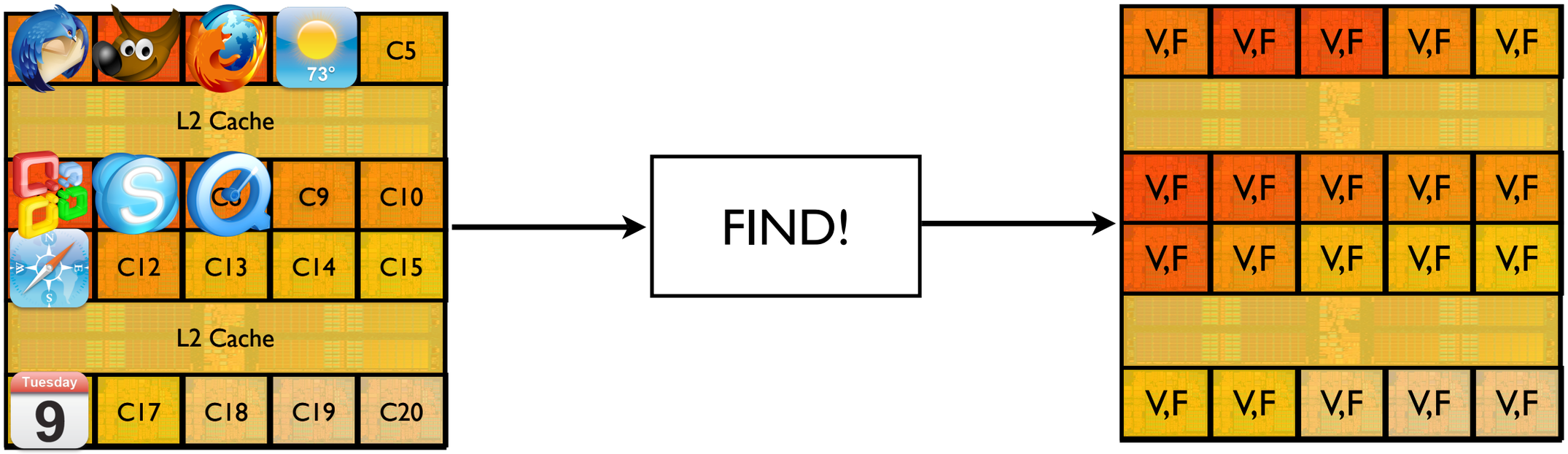
FIND!

best (V_i,F_i) of each core



Optimization problem

Given a mapping of threads to cores (variation-aware):

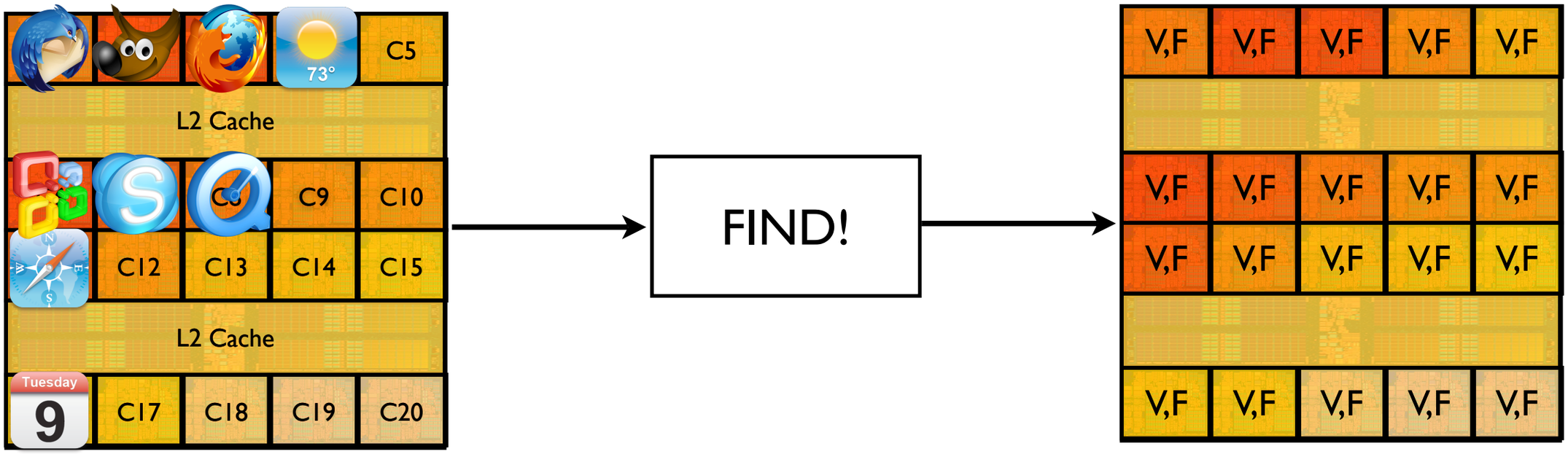


best (V_i,F_i) of each core

- **Goal:** maximize system throughput (MIPS)

Optimization problem

Given a mapping of threads to cores (variation-aware):



best (V_i,F_i) of each core

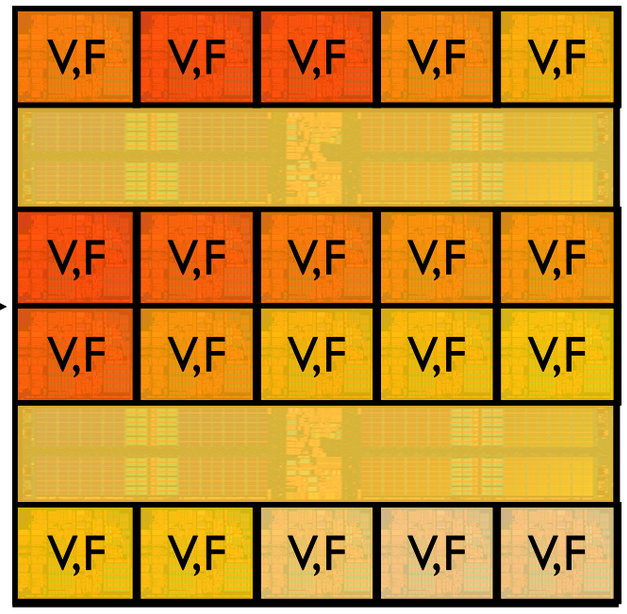
- **Goal:** maximize system throughput (MIPS)
- **Constraint:** keep total power below budget

Optimization problem

Given a mapping of threads to cores (variation-aware):

FIND!

best (V_i,F_i) of each core



- **Goal:** maximize system throughput (MIPS)
- **Constraint:** keep total power below budget

50W

75W

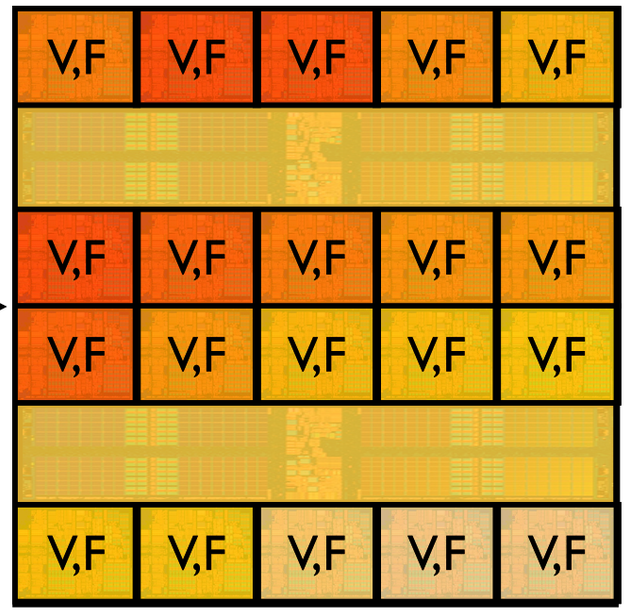
100W

Optimization problem

Given a mapping of threads to cores (variation-aware):

FIND!

best (V_i,F_i) of each core



- **Goal:** maximize system throughput (MIPS)
- **Constraint:** keep total power below budget
- **Runtime** system adaptation

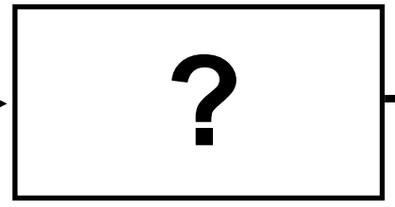
50W

75W

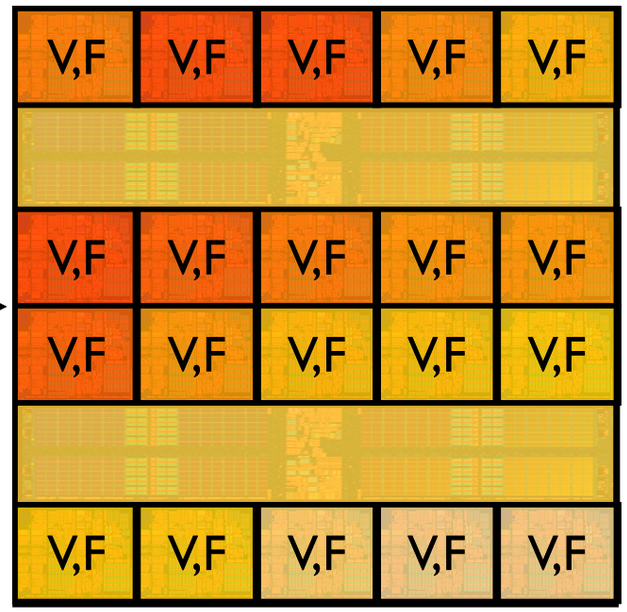
100W

Optimization problem

Given a mapping of threads to cores (variation-aware):



best (V_i,F_i) of each core



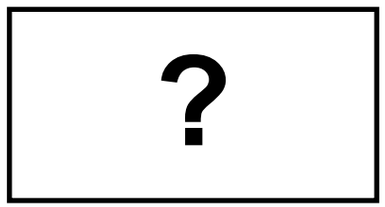
- **Goal:** maximize system throughput (MIPS)
- **Constraint:** keep total power below budget
- **Runtime** system adaptation

50W

75W

100W

Possible solutions



Possible solutions

?

Possible solutions

?

- Exhaustive search: too expensive

Possible solutions

?

- Exhaustive search: too expensive
- Simulated annealing (***SAnn***)
  - Not practical at runtime

Possible solutions

?

- Exhaustive search: too expensive
- Simulated annealing (***SAnn***)
  - Not practical at runtime
- Linear programming (***LinOpt***)
  - Simpler, faster
  - Requires some approximations

Possible solutions

LinOpt

- Exhaustive search: too expensive
- Simulated annealing (***SAnn***)
  - Not practical at runtime
- Linear programming (***LinOpt***)
  - Simpler, faster
  - Requires some approximations

Outline

- Variation-aware scheduling
- Variation-aware power management
  - Defining the optimization problem
  - Implementation
- Evaluation
- Conclusions

LinOpt problem definition

LinOpt problem definition

- **Linear programming:**

LinOpt problem definition

- **Linear programming:**
  - Maximize objective function: $f(x_1,...,x_n)$, with $x_1,...,x_n$ independent

LinOpt problem definition

- **Linear programming:**
  - Maximize objective function: $f(x_1,...,x_n)$, with $x_1,...,x_n$ independent
  - Subject to constraints such as: $g(x_1,...,x_n) < C$

LinOpt problem definition

- **Linear programming:**
  - Maximize objective function: $f(x_1,...,x_n)$, with $x_1,...,x_n$ independent
  - Subject to constraints such as: $g(x_1,...,x_n) < C$
  - f,g are linear functions

LinOpt problem definition

- **Linear programming:**
  - Maximize objective function: $f(x_1,...,x_n)$, with $x_1,...,x_n$ independent
  - Subject to constraints such as: $g(x_1,...,x_n) < C$
  - f,g are linear functions
- **Unknowns:** voltages $V_1,...,V_n$ for all cores

LinOpt problem definition

- **Linear programming:**
  - Maximize objective function: $f(x_1,...,x_n)$, with $x_1,...,x_n$ independent
  - Subject to constraints such as: $g(x_1,...,x_n) < C$
  - f,g are linear functions
- **Unknowns:** voltages $V_1,...,V_n$ for all cores
- **Objective function:** maximize CMP throughput

LinOpt problem definition

- **Linear programming:**
  - Maximize objective function: $f(x_1,...,x_n)$, with $x_1,...,x_n$ independent
  - Subject to constraints such as: $g(x_1,...,x_n) < C$
  - *f,g* are linear functions
- **Unknowns:** voltages $V_1,...,V_n$ for all cores
- **Objective function:** maximize CMP throughput
  - Throughput (MIPS) = Frequency X IPC = $f(V_1,...,V_n)$

LinOpt problem definition

- **Linear programming:**
  - Maximize objective function: $f(x_1,...,x_n)$, with $x_1,...,x_n$ independent
  - Subject to constraints such as: $g(x_1,...,x_n) < C$
  - f,g are linear functions
- **Unknowns:** voltages $V_1,...,V_n$ for all cores
- **Objective function:** maximize CMP throughput
  - Throughput (MIPS) = Frequency X IPC = $f(V_1,...,V_n)$
- **Constraint:** keep power under P_{target}

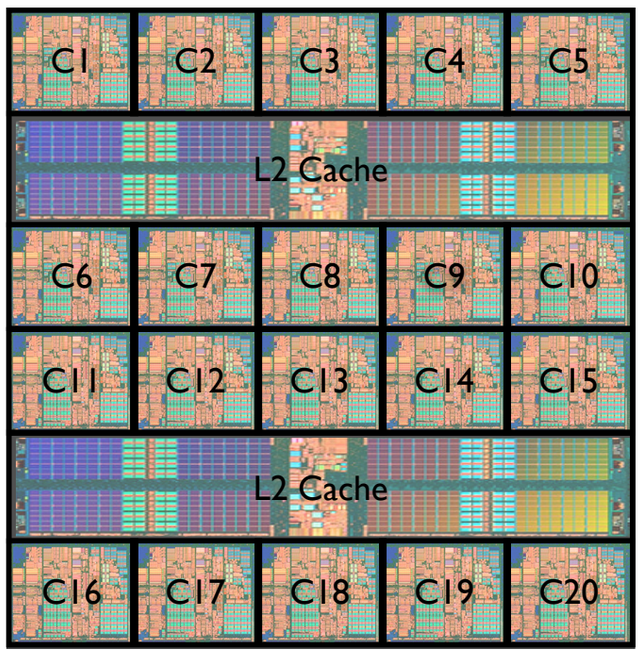
LinOpt problem definition

- **Linear programming:**
  - Maximize objective function: $f(x_1,...,x_n)$, with $x_1,...,x_n$ independent
  - Subject to constraints such as: $g(x_1,...,x_n) < C$
  - f,g are linear functions
- **Unknowns:** voltages $V_1,...,V_n$ for all cores
- **Objective function:** maximize CMP throughput
  - Throughput (MIPS) = Frequency X IPC = $f(V_1,...,V_n)$
- **Constraint:** keep power under P_{target}
  - Power = g(V)

LinOpt implementation

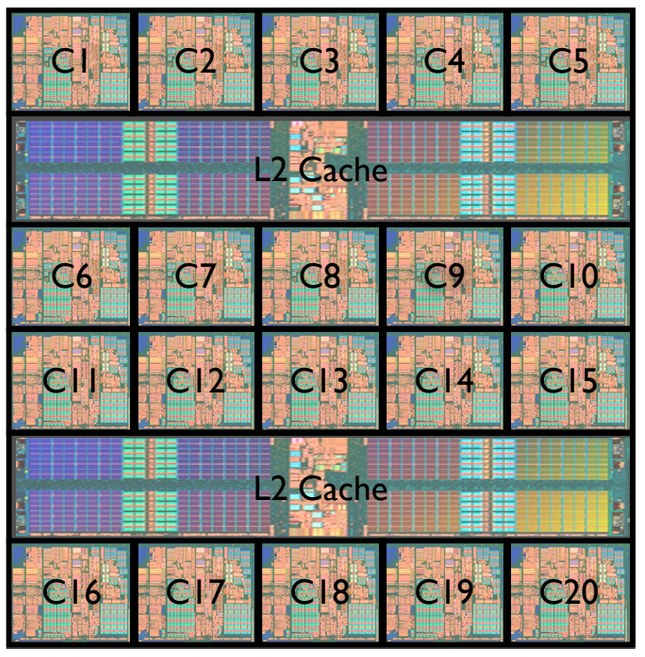
LinOpt implementation

- ***LinOpt*** works together with the OS scheduler
  - OS scheduler maps applications to cores (e.g. ***VarF&AppIPC***)
  - ***LinOpt*** then finds *(V,F)* settings for each core



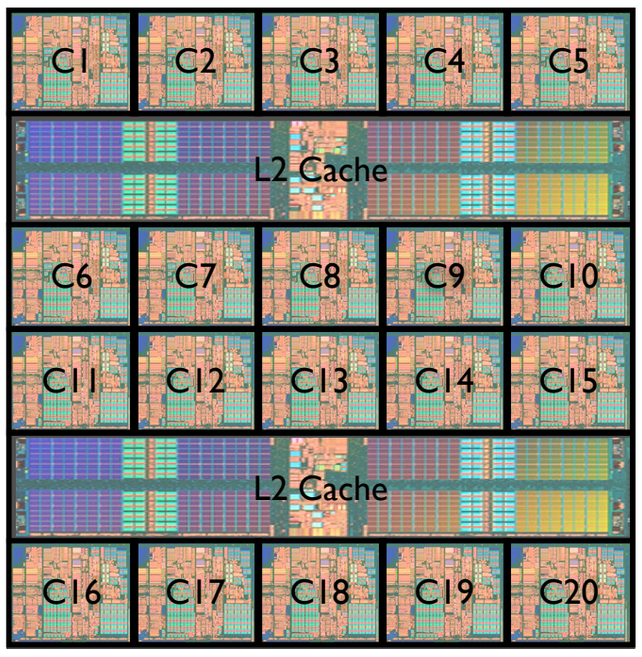
LinOpt implementation

- ***LinOpt*** works together with the OS scheduler
  - OS scheduler maps applications to cores (e.g. ***VarF&AppIPC***)
  - ***LinOpt*** then finds *(V,F)* settings for each core
- ***LinOpt*** runs periodically as a system process



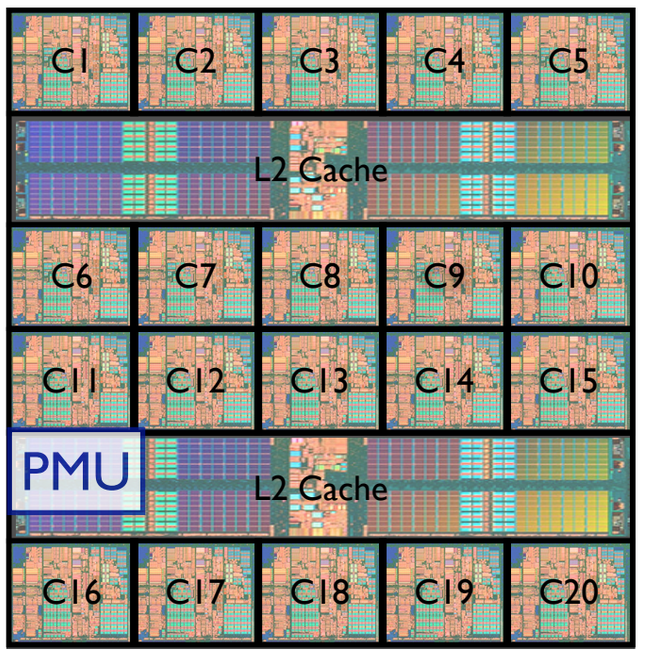
LinOpt implementation

- ***LinOpt*** works together with the OS scheduler
  - OS scheduler maps applications to cores (e.g. ***VarF&AppIPC***)
  - ***LinOpt*** then finds *(V,F)* settings for each core
- ***LinOpt*** runs periodically as a system process
  - On a core



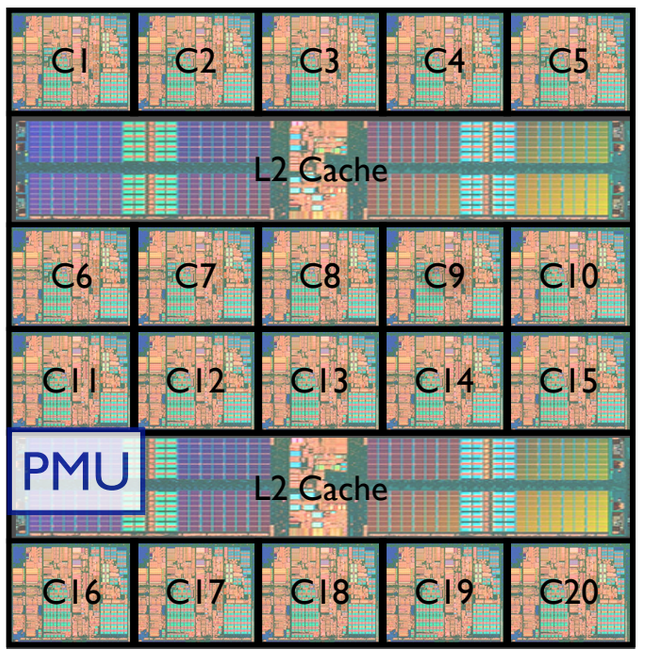
LinOpt implementation

- ***LinOpt*** works together with the OS scheduler
  - OS scheduler maps applications to cores (e.g. ***VarF&AppIPC***)
  - ***LinOpt*** then finds *(V,F)* settings for each core
- ***LinOpt*** runs periodically as a system process
  - On a core
  - Power management unit (PMU) - e.g., *Foxton*



LinOpt implementation

- ***LinOpt*** works together with the OS scheduler
  - OS scheduler maps applications to cores (e.g. ***VarF&AppIPC***)
  - ***LinOpt*** then finds *(V,F)* settings for each core
- ***LinOpt*** runs periodically as a system process
  - On a core
  - Power management unit (PMU) - e.g., *Foxton*
- ***LinOpt*** uses profile information as input



LinOpt implementation

LinOpt implementation

Post-manufacturing profiling

Each core: frequency, static power

LinOpt implementation

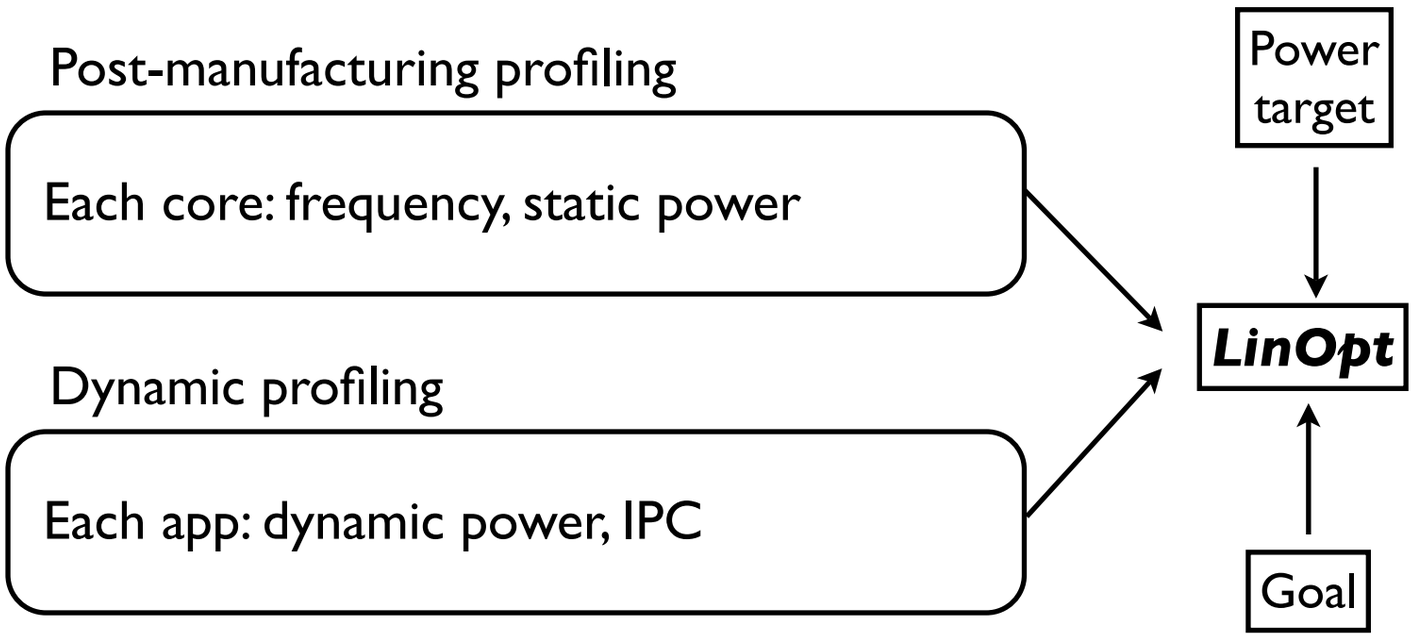
Post-manufacturing profiling

Each core: frequency, static power

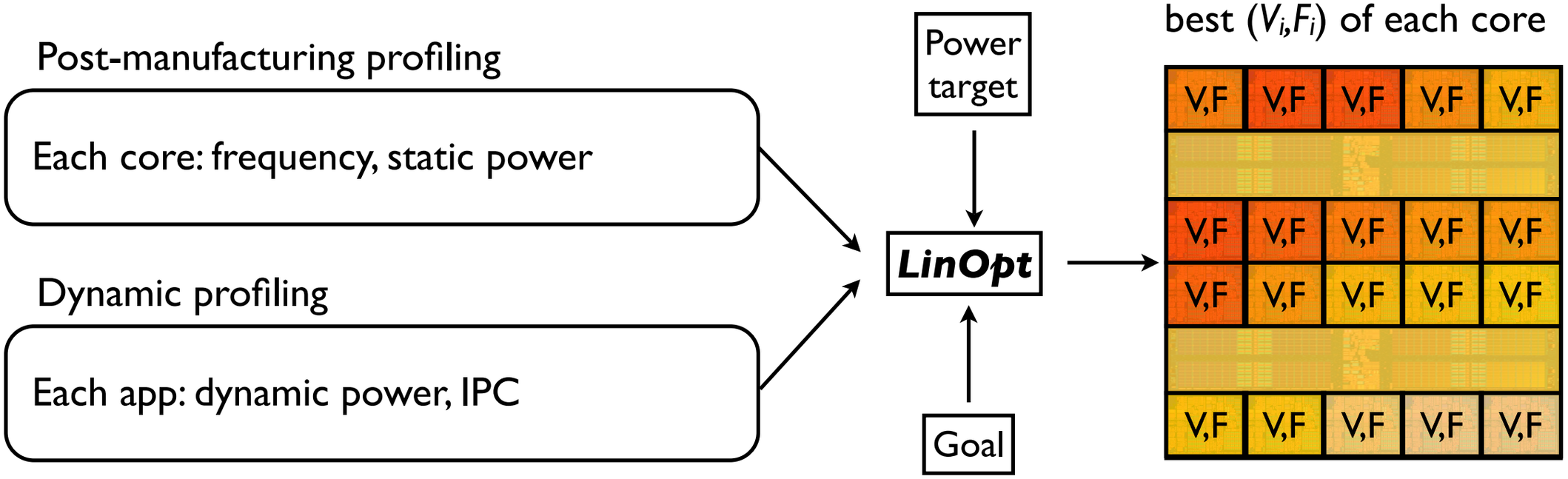
Dynamic profiling

Each app: dynamic power, IPC

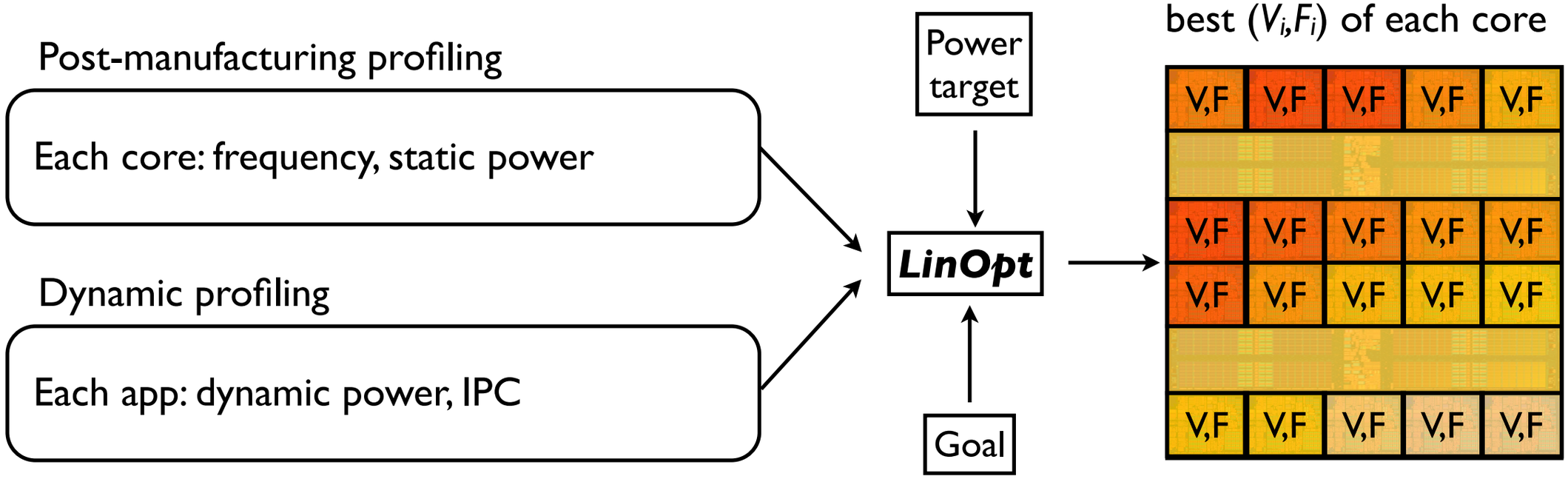
LinOpt implementation

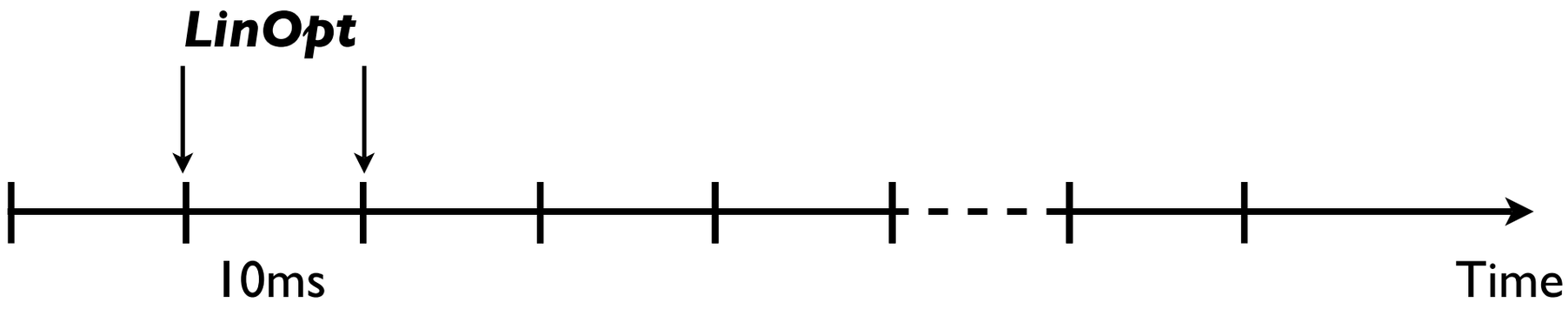


LinOpt implementation

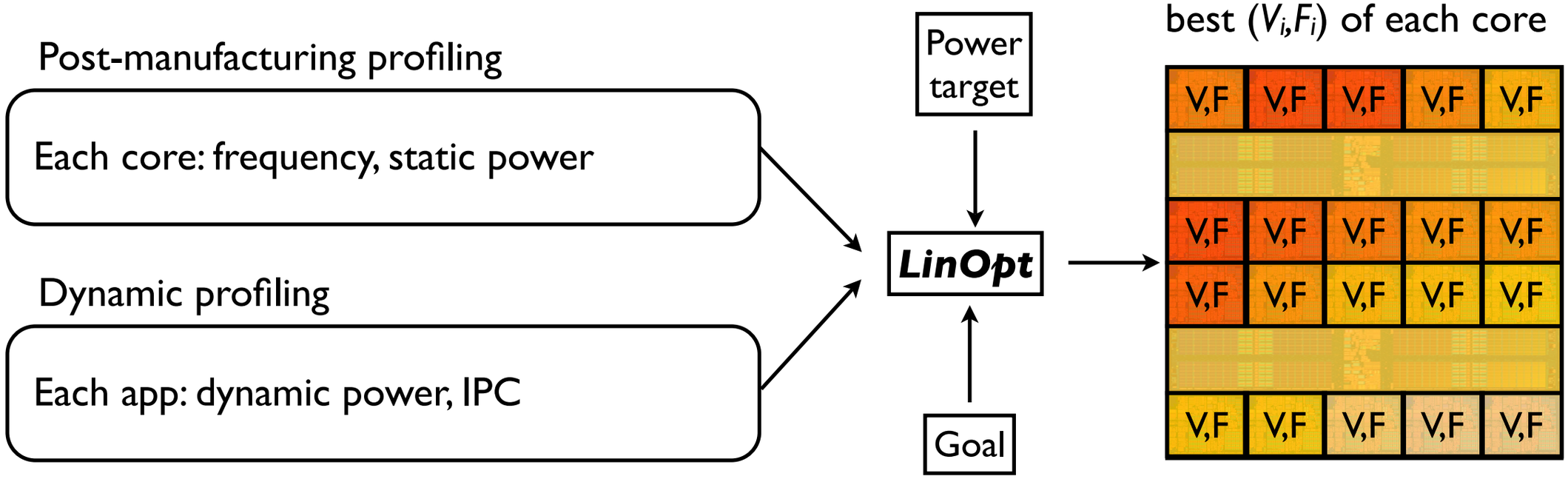


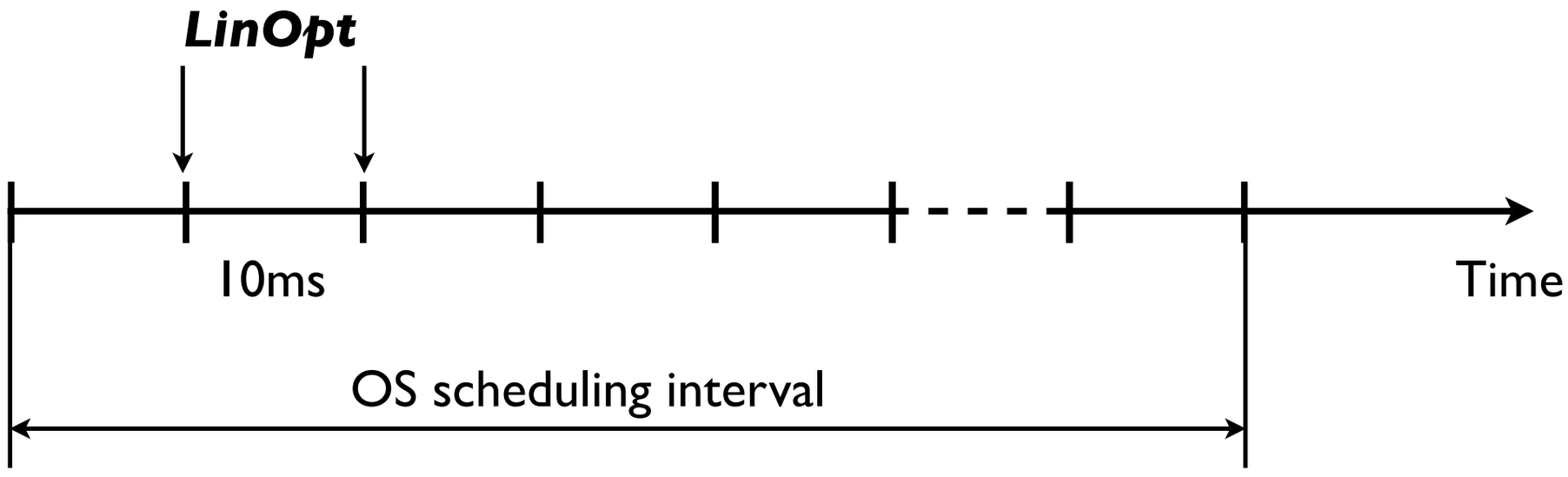
LinOpt implementation





LinOpt implementation





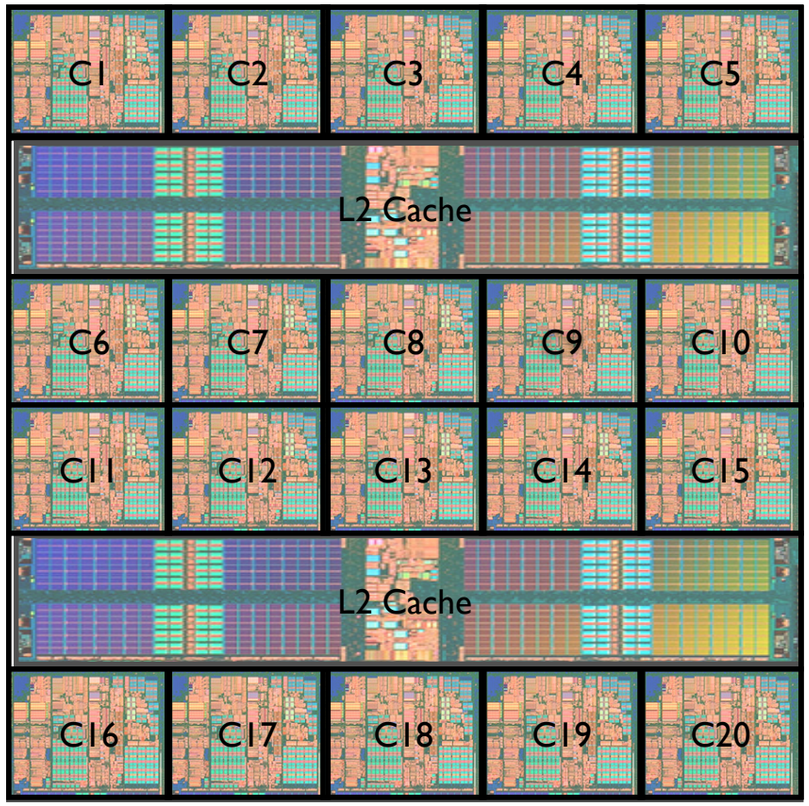
Outline

- Variation-aware scheduling
- Variation-aware power management
  - Defining the optimization problem
  - Implementation
- Evaluation
- Conclusions

Evaluation infrastructure

- Process variation model - *VARIUS* [IEEE TSM'08]
  - Monte Carlo simulations for 200 chips
- SESC - cycle accurate microarchitectural simulator
- SPICE model - leakage power
- Hotspot - temperature estimation

Evaluation infrastructure



- 20-core CMP
  - 2-issue, OOO cores
  - Shared L2 cache
- 32nm technology, 4GHz

- Multiprogrammed workload:
  - From a pool of SPECint and SPECfp benchmarks

Variation-aware scheduling

Variation-aware scheduling

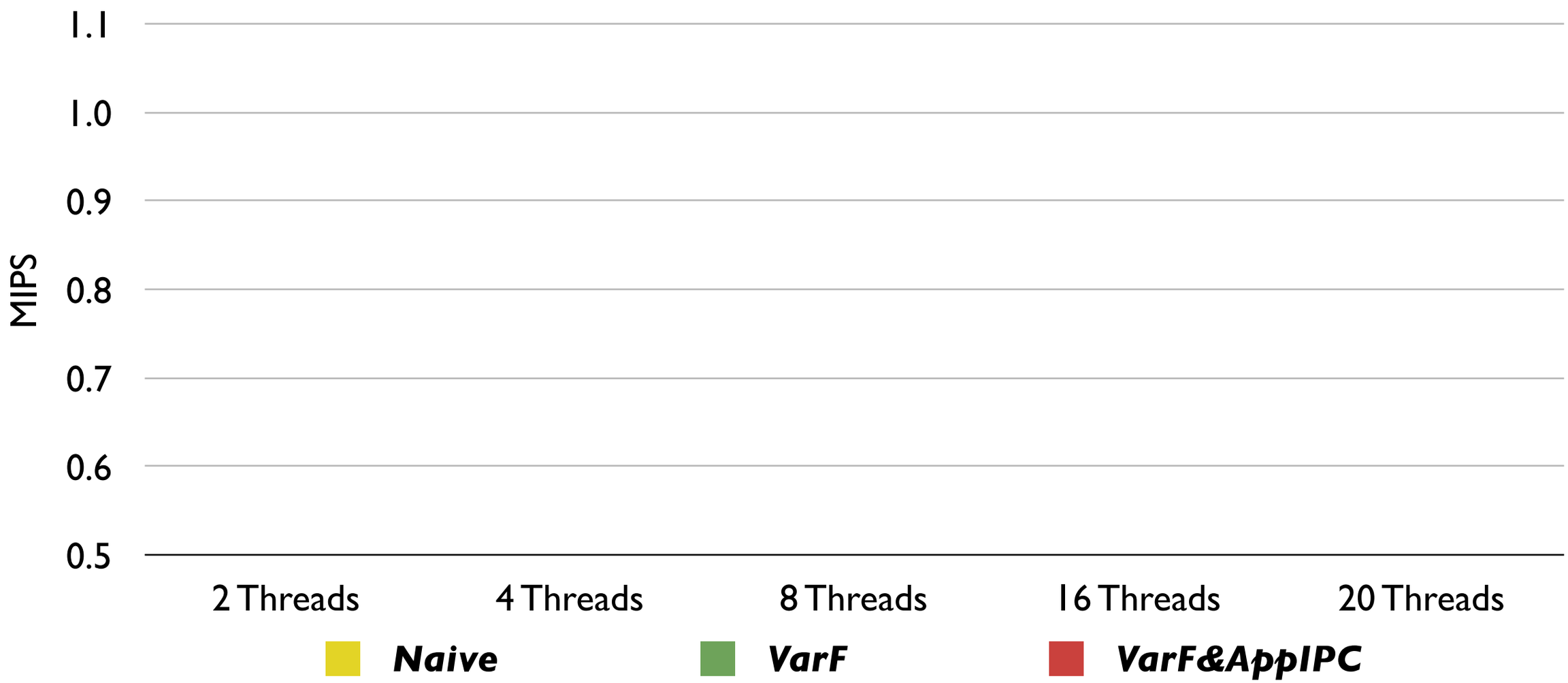
Goal: Improve CMP throughput

Variation-aware scheduling

Goal: Improve CMP throughput

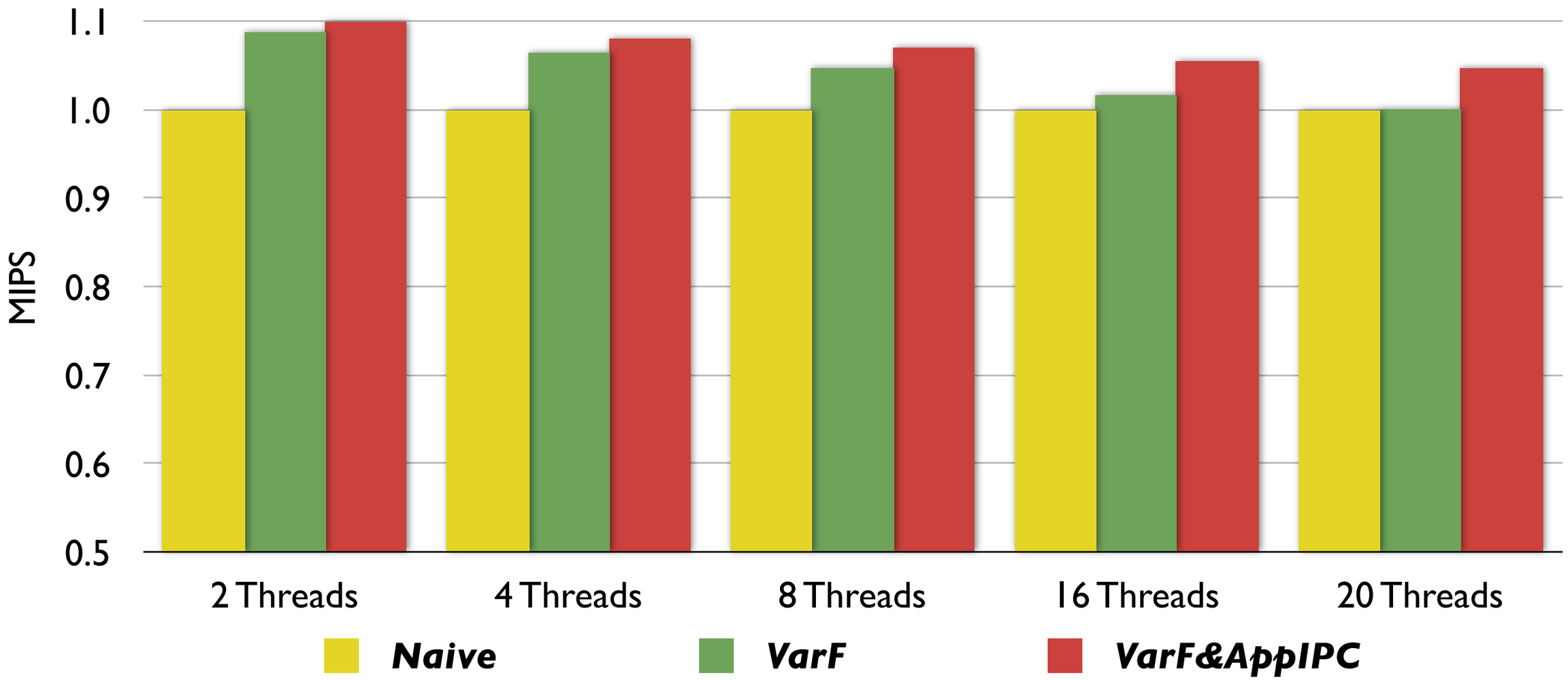
Variation-aware scheduling

Goal: Improve CMP throughput



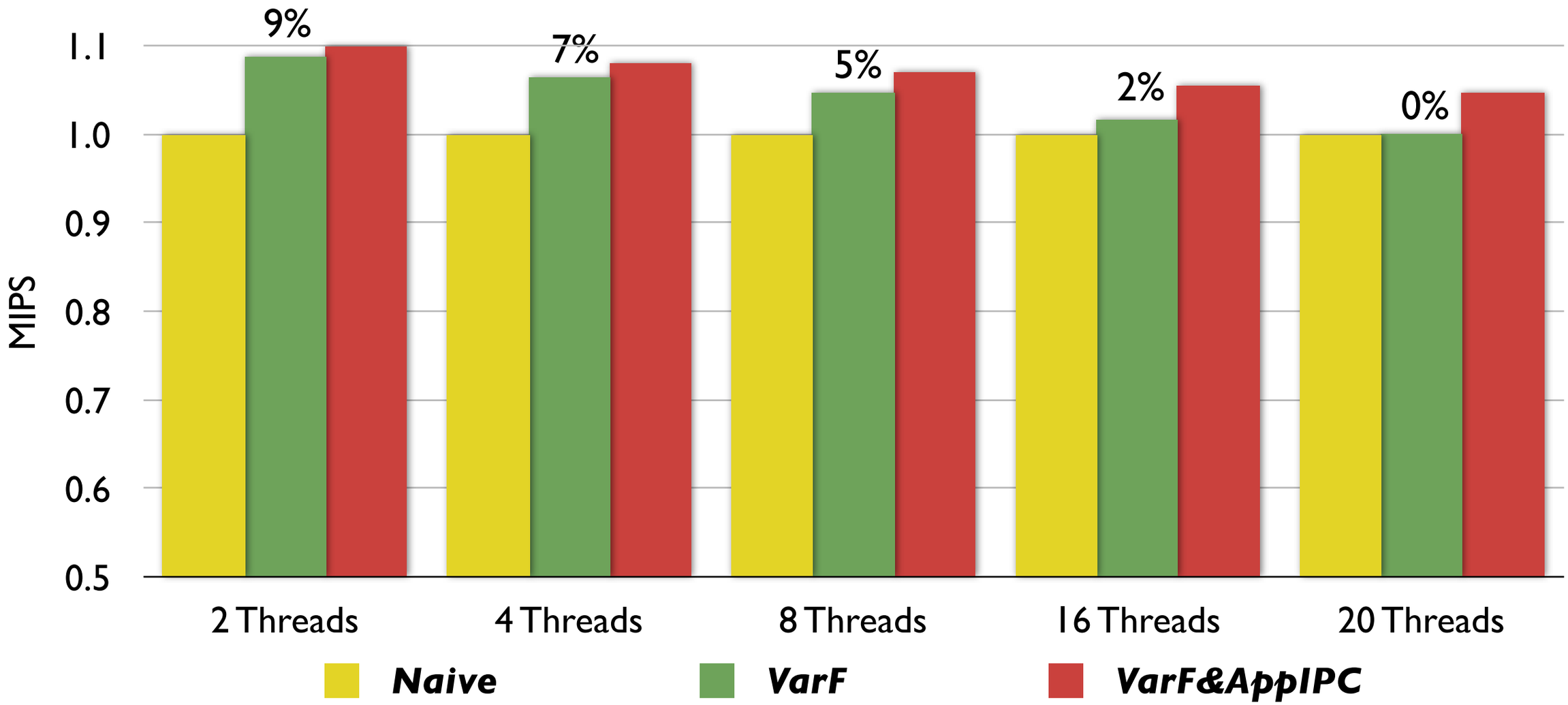
Variation-aware scheduling

Goal: Improve CMP throughput



Variation-aware scheduling

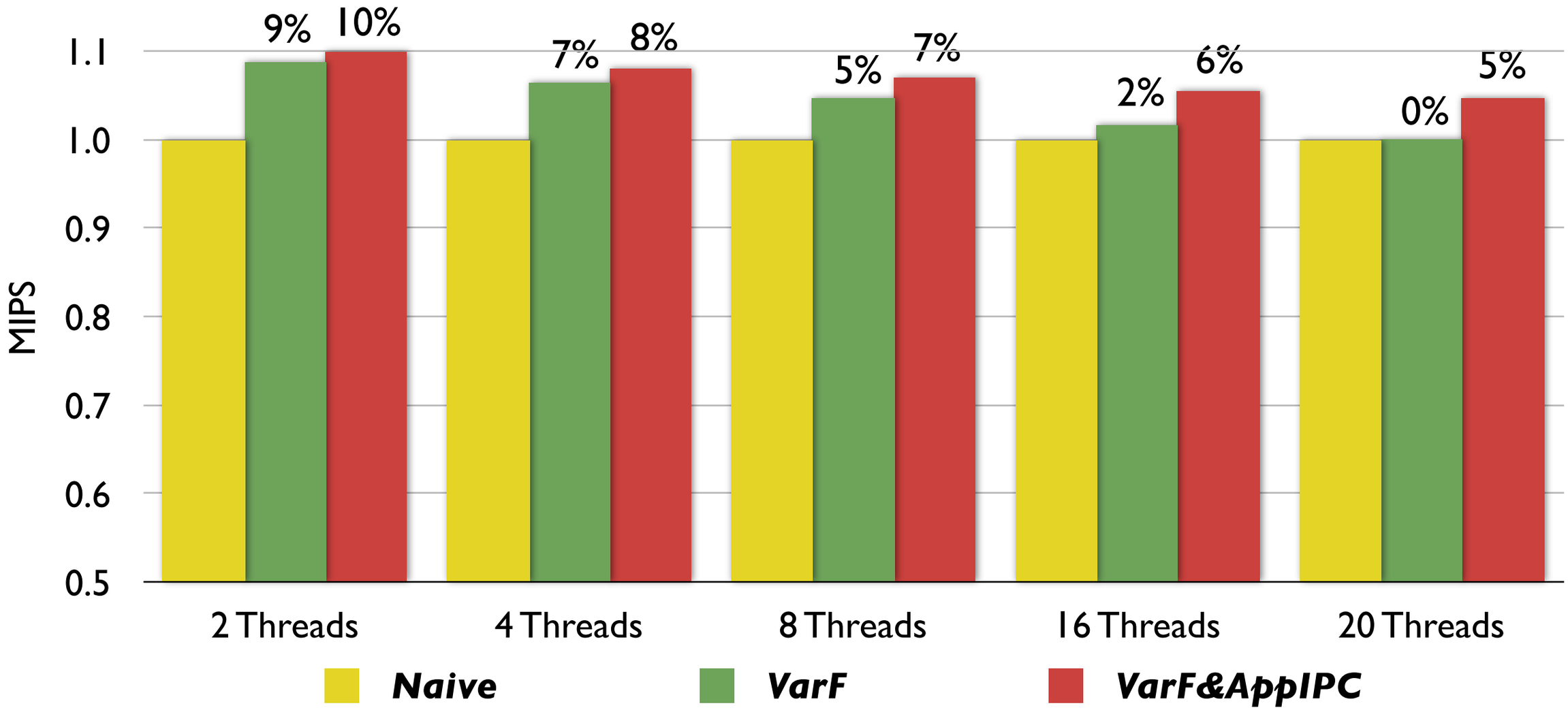
Goal: Improve CMP throughput



- ***VarF*:** up to 9% throughput improvement over Naive

Variation-aware scheduling

Goal: Improve CMP throughput



- ***VarF*:** up to 9% throughput improvement over Naive
- ***VarF&AppIPC*** scales better with number of threads: 5-10% improvement over Naive

Variation-aware power management

Global power management algorithms:

- Goal: maximize throughput
- Constraint: keep power below budget (75W)

Variation-aware power management

Global power management algorithms:

- Goal: maximize throughput
- Constraint: keep power below budget (75W)

Foxton+: baseline

Variation-aware power management

Global power management algorithms:

- Goal: maximize throughput
- Constraint: keep power below budget (75W)

Foxton+: baseline

LinOpt: proposed scheme

Variation-aware power management

Global power management algorithms:

- Goal: maximize throughput
- Constraint: keep power below budget (75W)

Foxton+: baseline

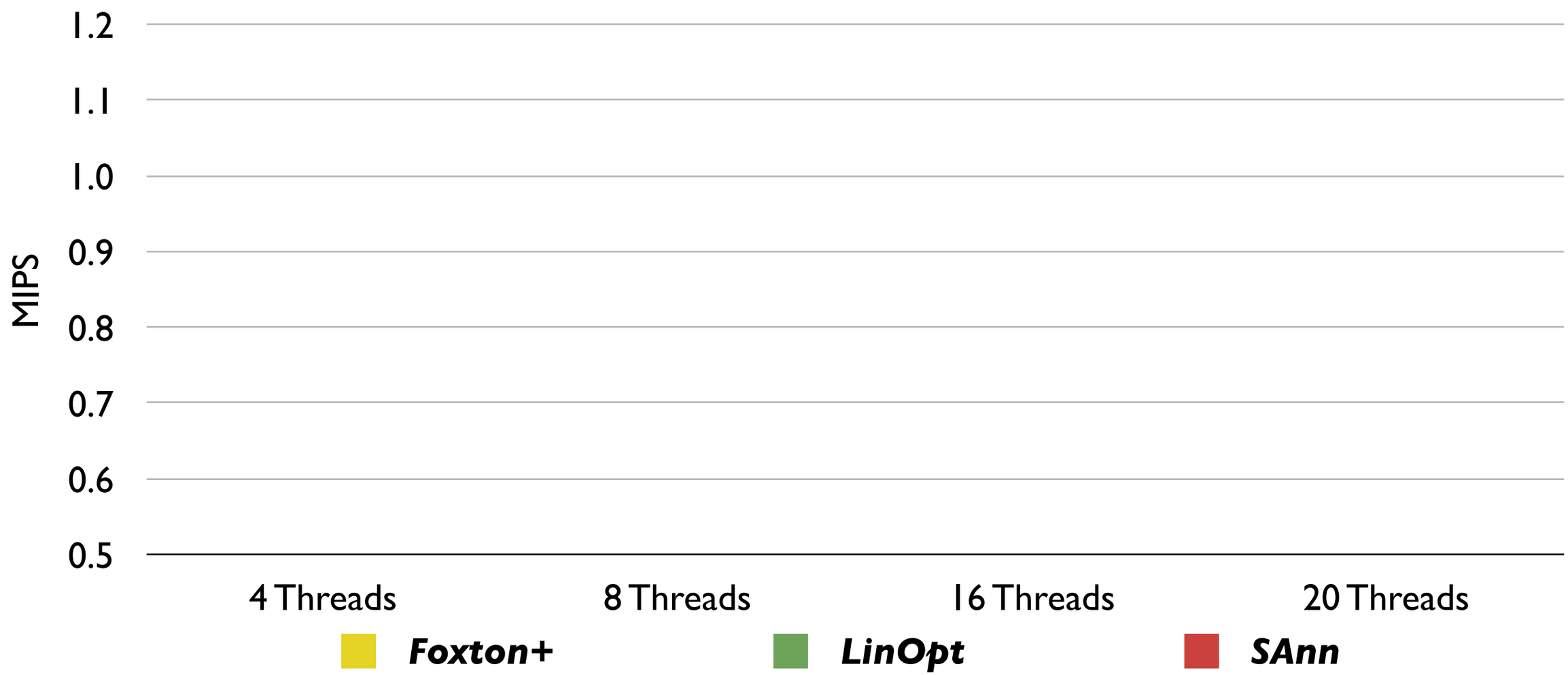
LinOpt: proposed scheme

SAnn: approximate upper bound

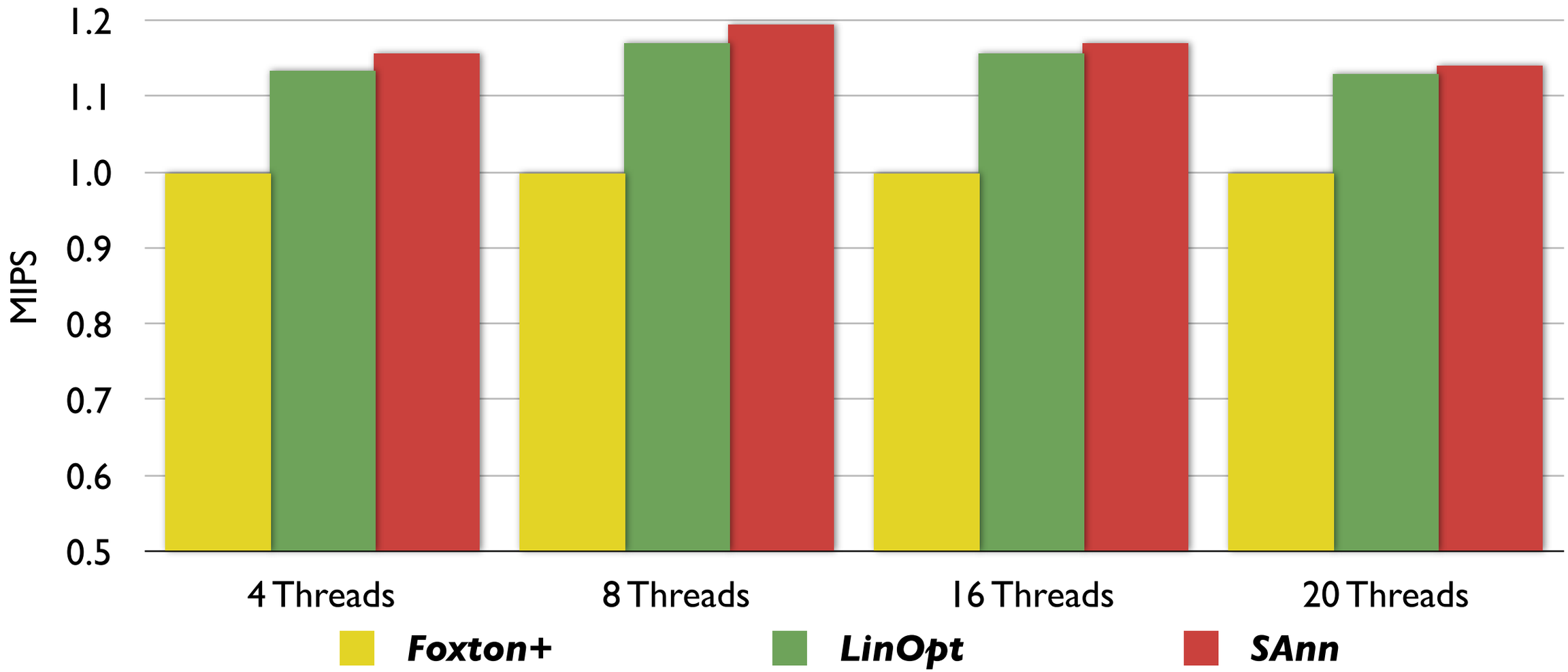
Variation-aware power management

Foxton+ LinOpt SAnn

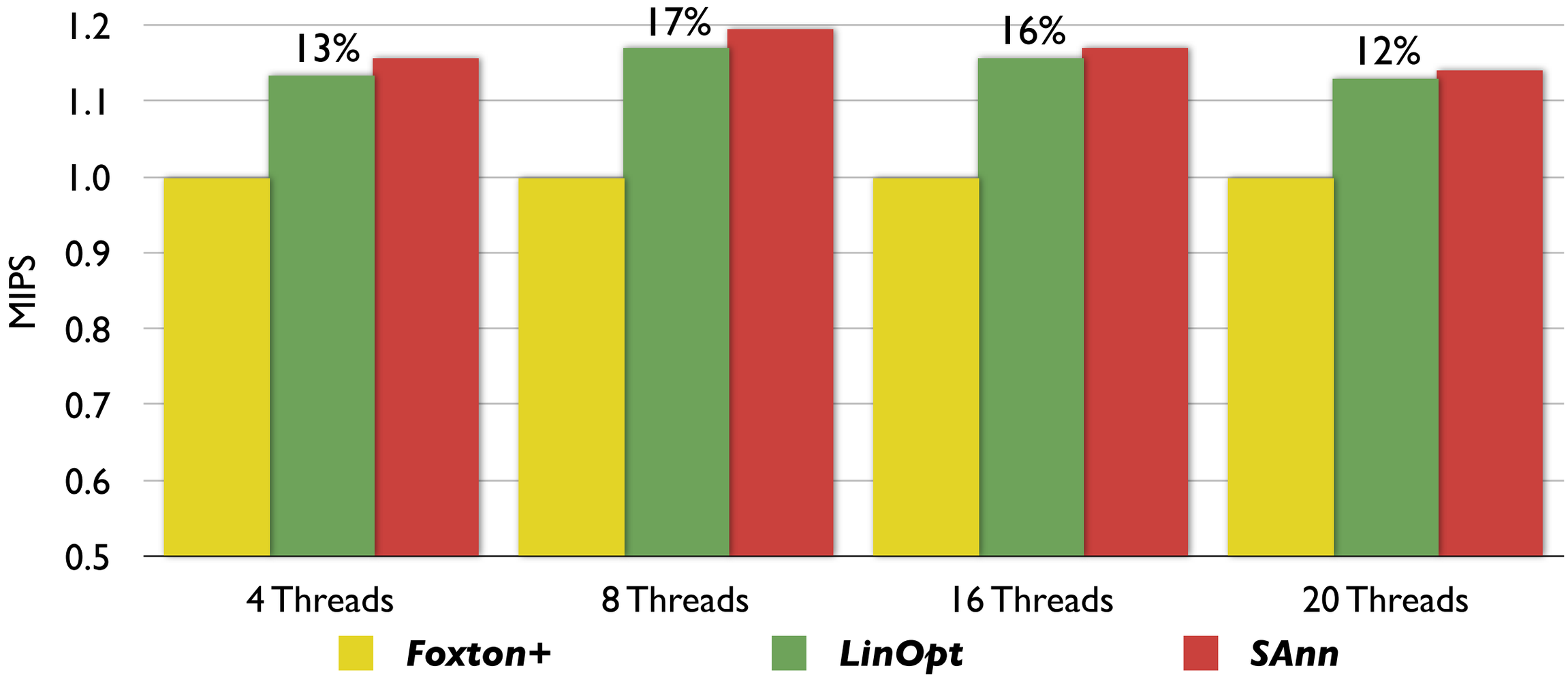
Variation-aware power management



Variation-aware power management

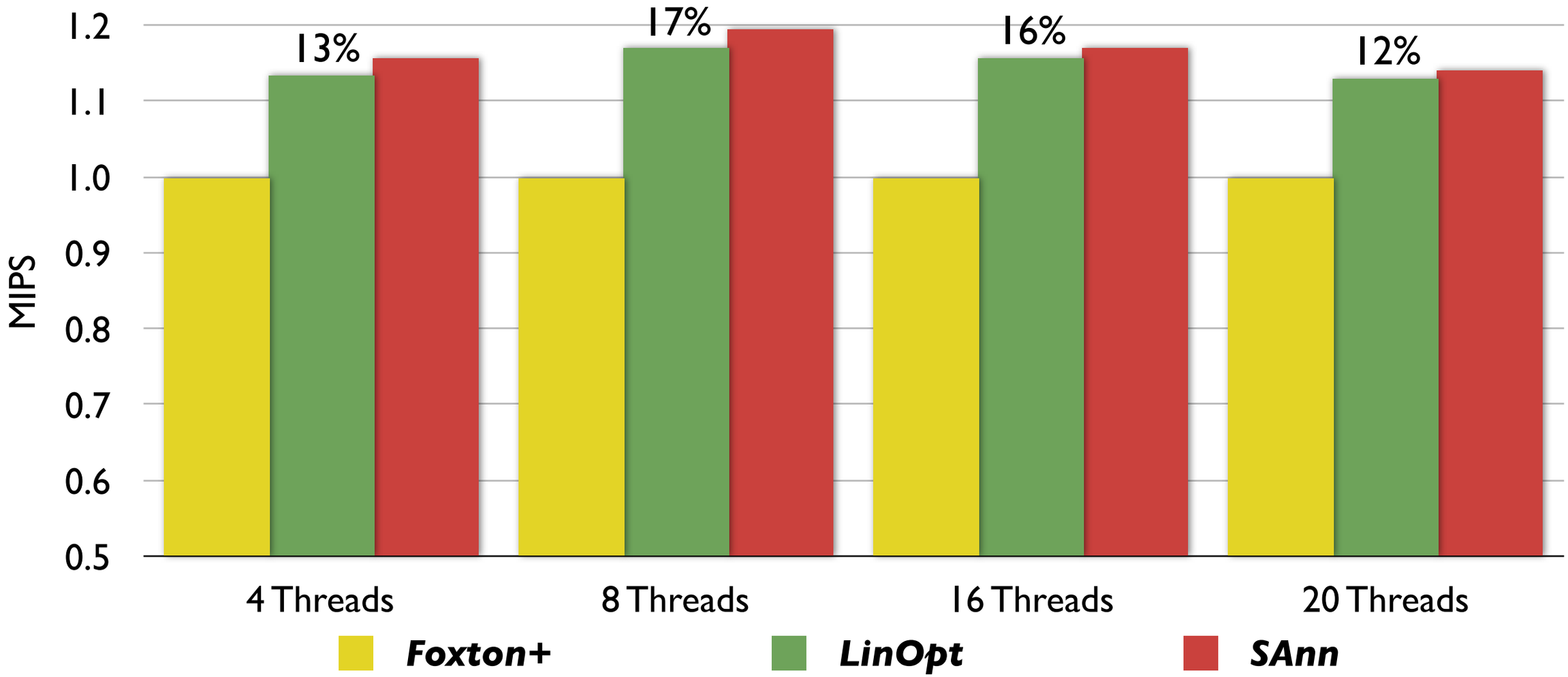


Variation-aware power management



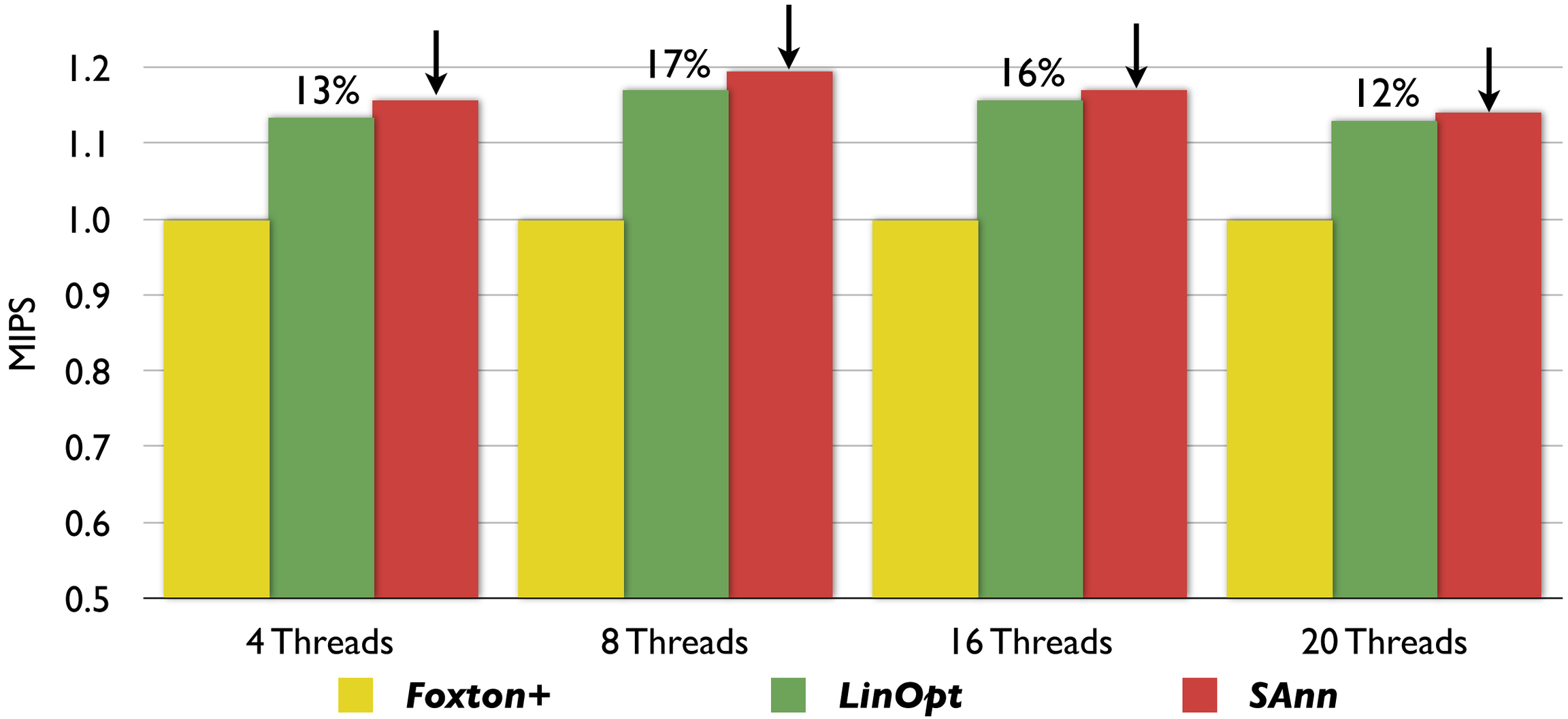
- ***LinOpt:*** 12-17% improvement over ***Foxton+***, at the same power

Variation-aware power management



- ***LinOpt:*** 12-17% improvement over ***Foxton+***, at the same power
  - 30-38% reduction in ED^2

Variation-aware power management



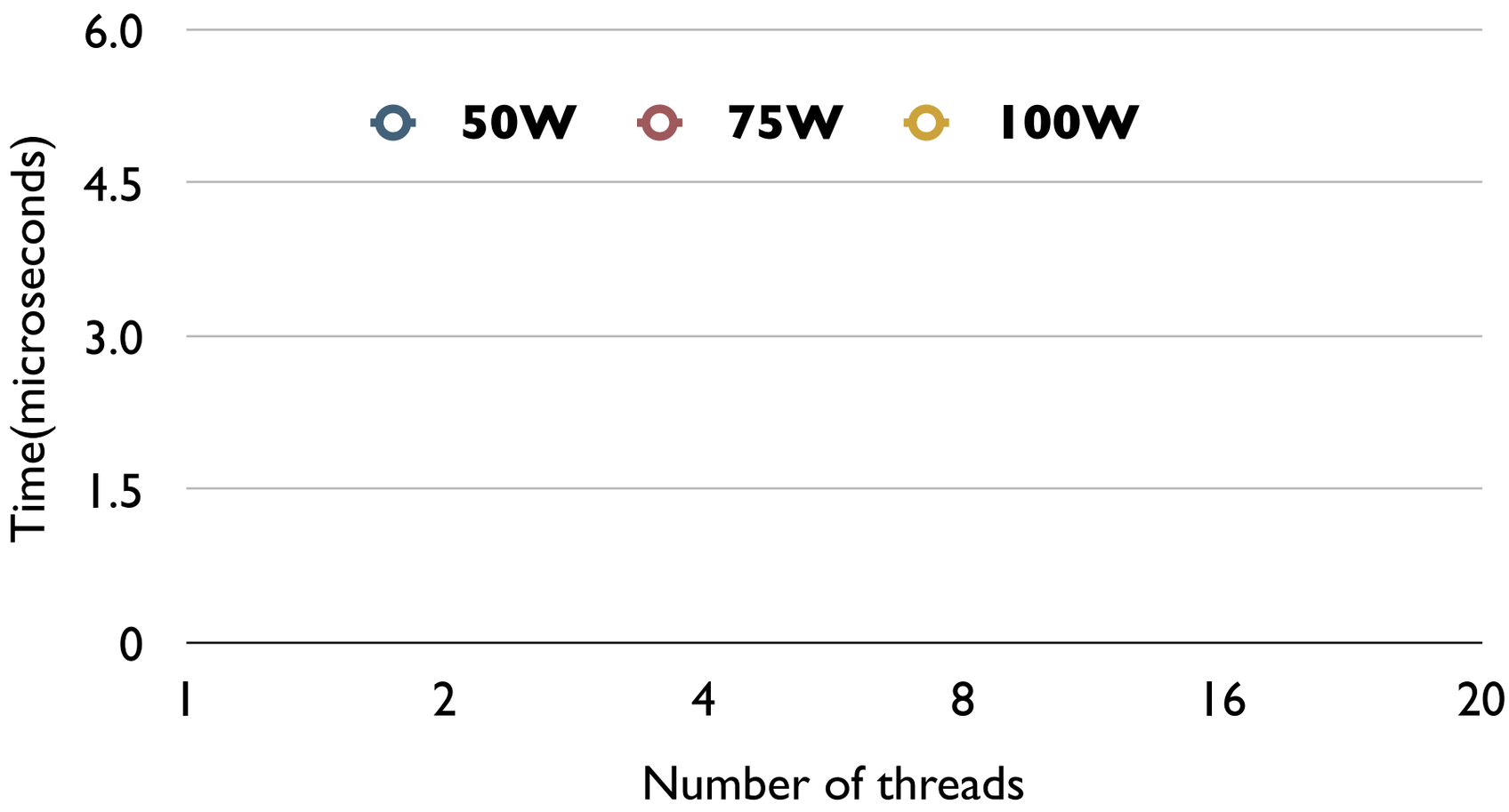
- ***LinOpt:*** 12-17% improvement over ***Foxton+***, at the same power
  - 30-38% reduction in ED^2
- ***LinOpt*** within 2% of ***SAnn***

Time overhead of ***LinOpt***

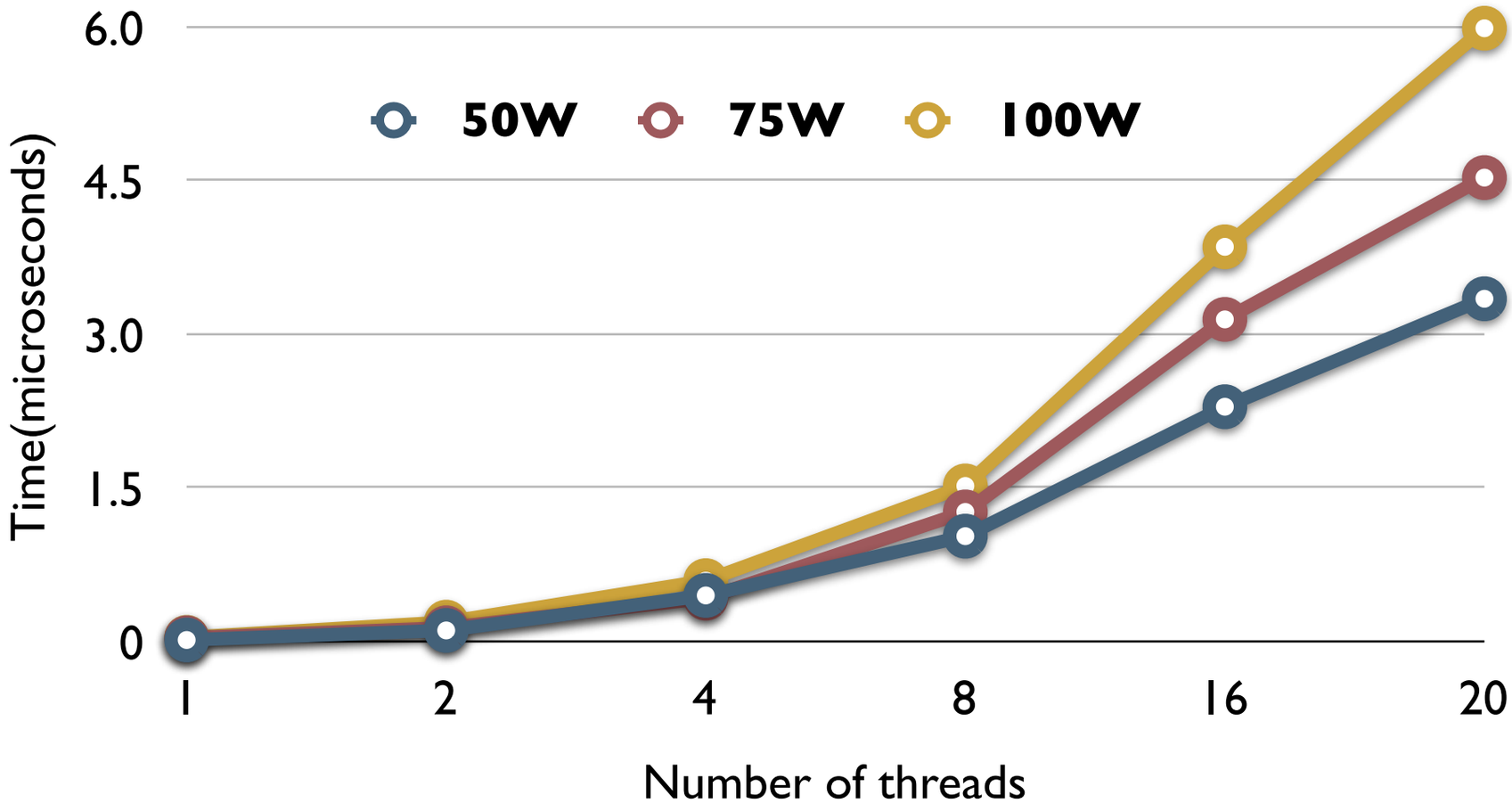
Time overhead of ***LinOpt***

50W 75W 100W

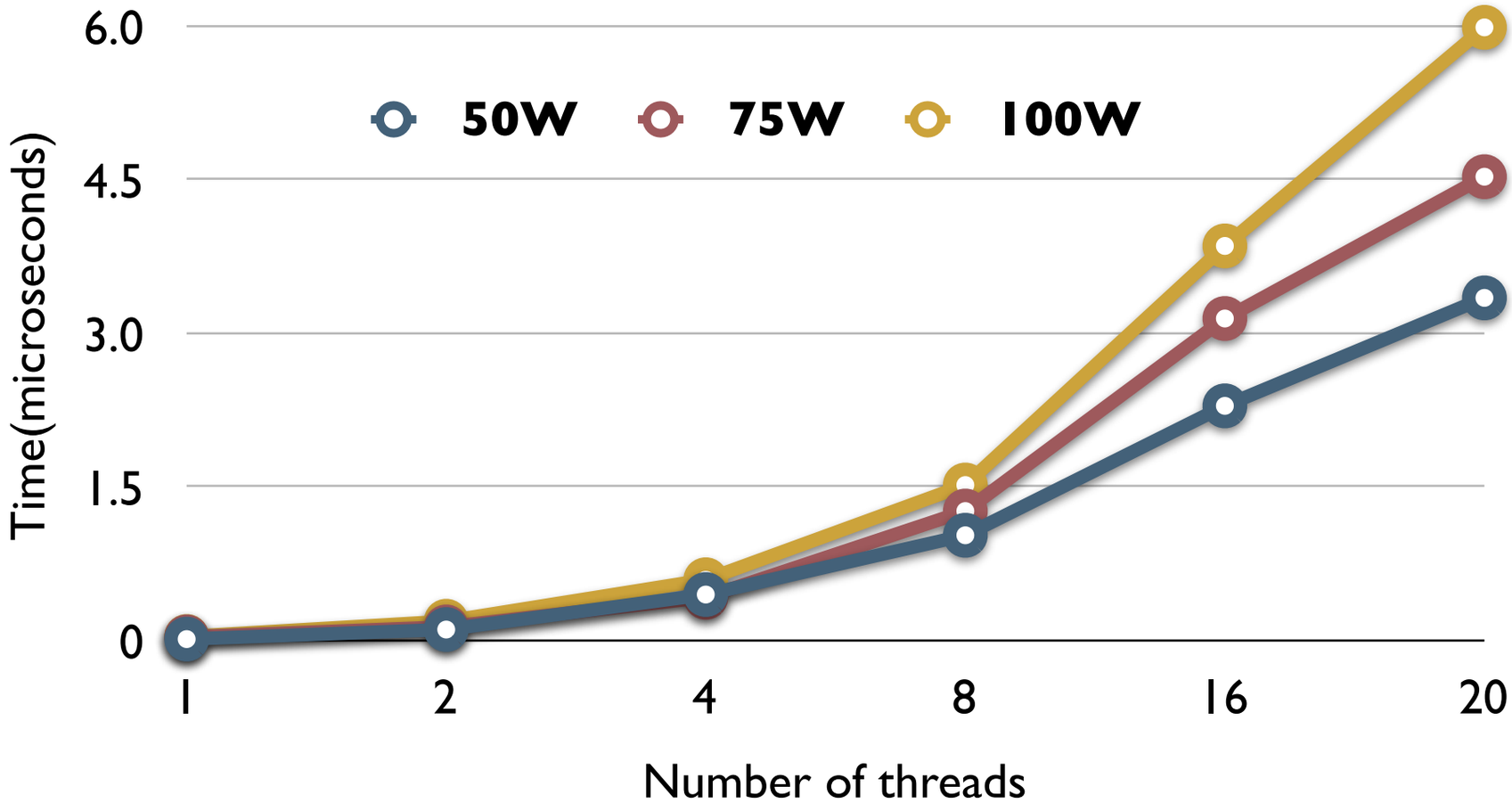
Time overhead of ***LinOpt***



Time overhead of ***LinOpt***

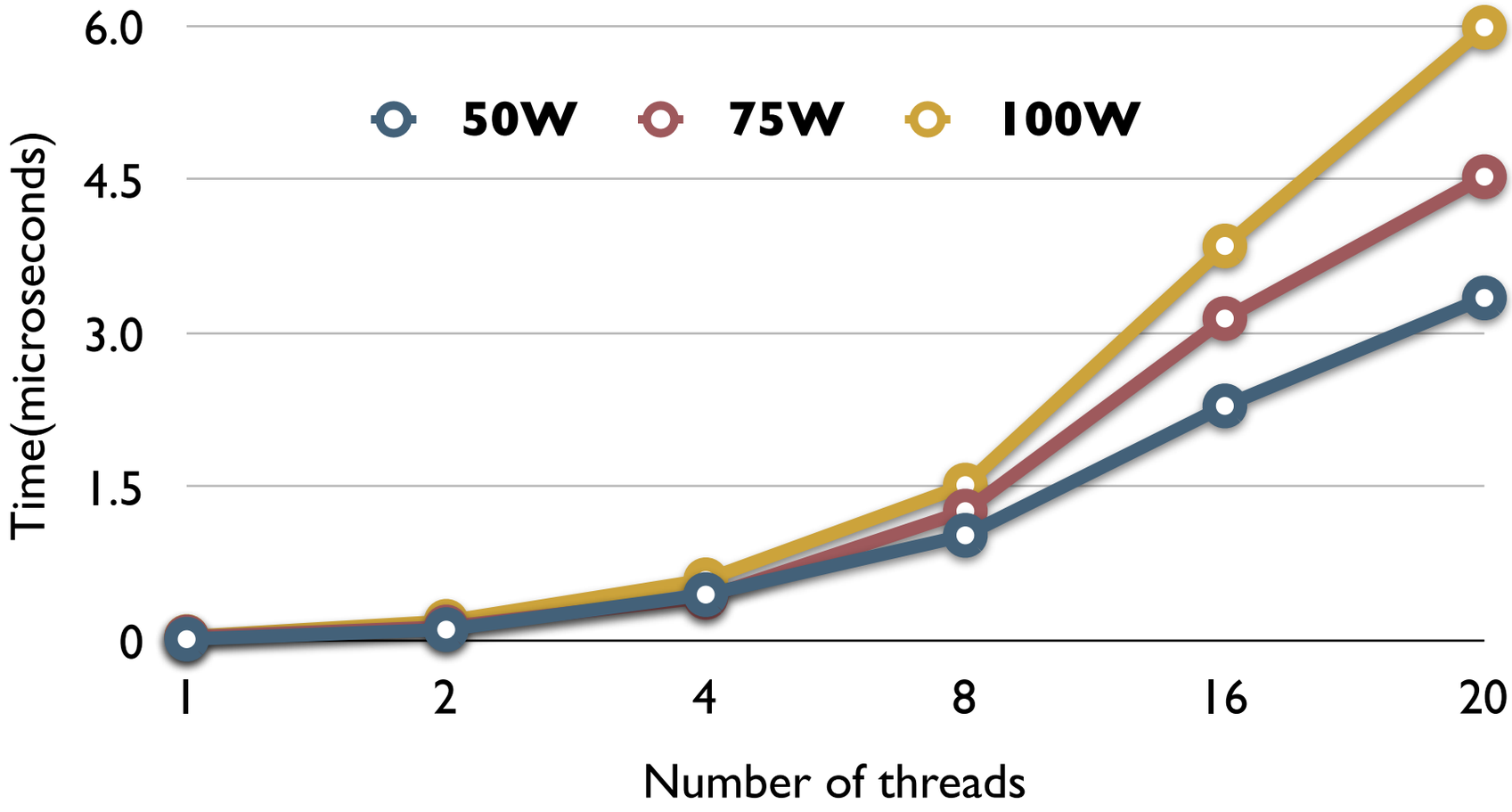


Time overhead of ***LinOpt***



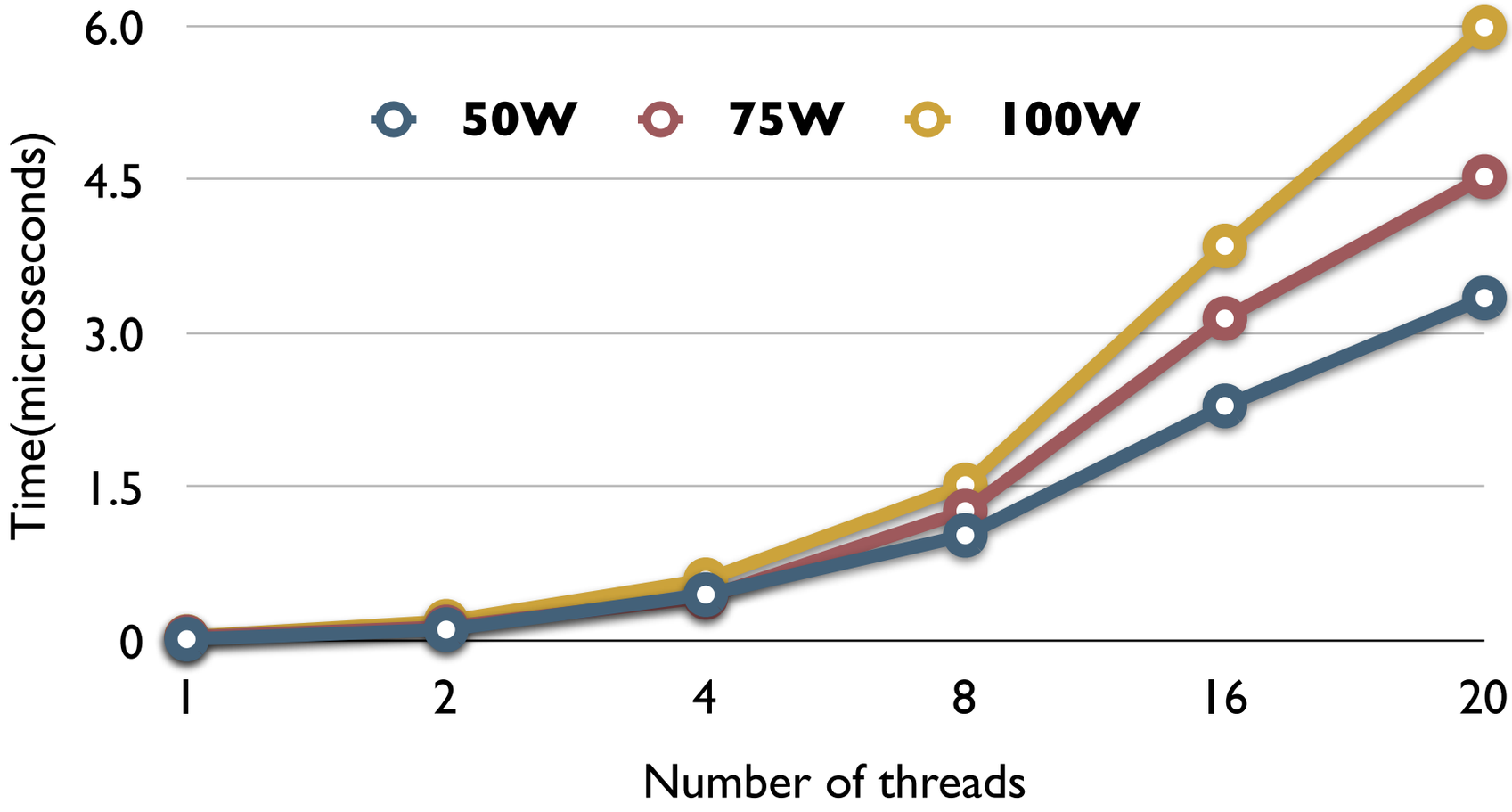
- Low overhead even for large problem size

Time overhead of ***LinOpt***



- Low overhead even for large problem size
  - Up to 6 μs for 20 threads

Time overhead of ***LinOpt***



- Low overhead even for large problem size
  - Up to 6 µs for 20 threads
  - ***LinOpt*** runs on a core every 1-10 ms - negligible impact

Conclusions

- We showed the value of exposing variation in core frequency and power to the OS
- Proposed a set of scheduling algorithms
  - reduce CMP power consumption (2-16%)
  - improve CMP throughput (5-10%)
- Proposed a power management algorithm
  - improve CMP throughput for a given power budget (12-17%)

Variation Aware Application Scheduling and Power Management for Chip Multiprocessors

Radu Teodorescu* and Josep Torrellas

Computer Science Department
University of Illinois at Urbana-Champaign
http://iacoma.cs.uiuc.edu

*now at Ohio State University

i-acoma group