Computer Systems: (Parallel) Computer Architecture

Presenter: Sandeep K. S. Gupta

Reference:

- D. Patterson Slides
- M. Hill Slides

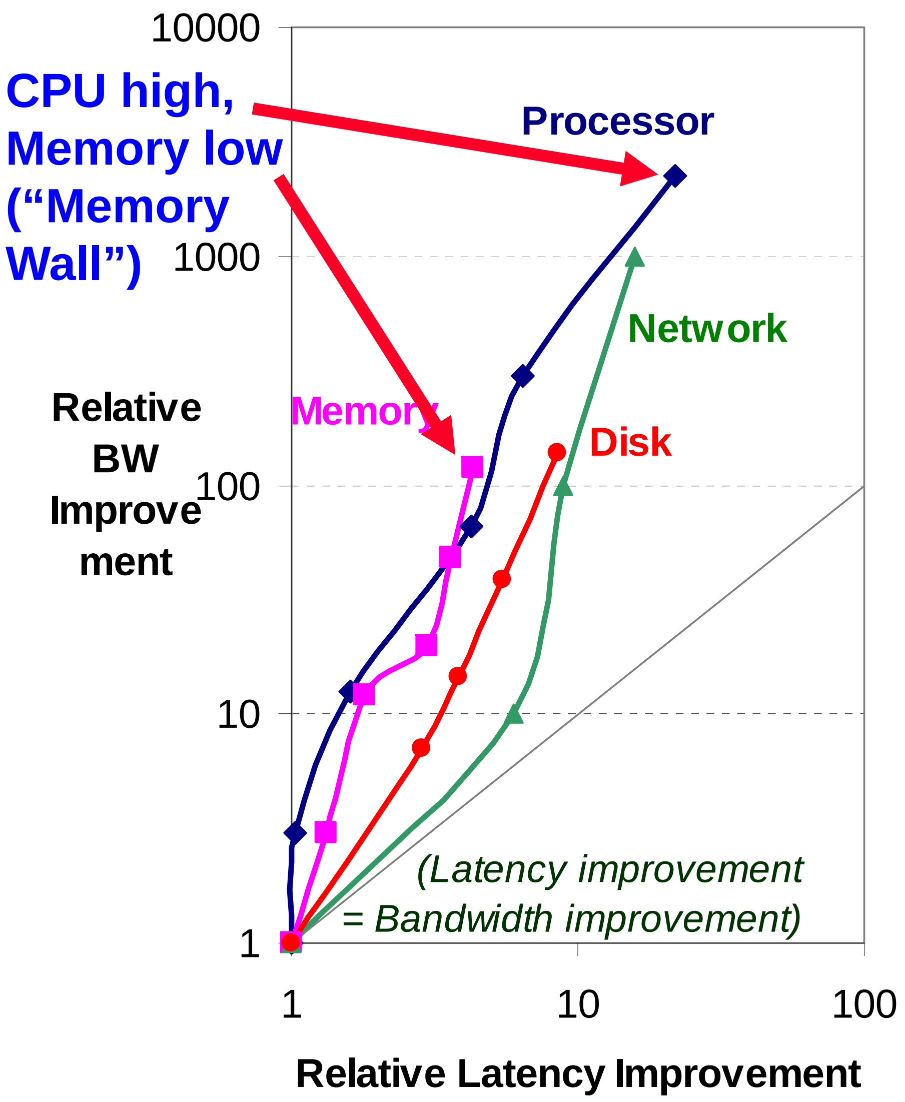
Arizona State University

School of computing, informatics, and decision systems engineering

Agenda

- Recap: Latency Lacks Behind Bandwidth
- Some Solutions
- Amdahl's Law for Multi-cores: Case for Dynamic Architecture
- Amdahl’s Law Energy-Efficiency

Latency Lags Bandwidth and Memory Wall



6 Reasons Latency Lags Bandwidth

1. Moore’s Law helps BW more than latency
2. Distance limits latency
3. Bandwidth easier to sell (“bigger=better”)
4. Latency helps BW, but not vice versa
5. Bandwidth hurts latency
6. Operating System overhead hurts Latency more than Bandwidth

Summary of Technology Trends

- **For disk, LAN, memory, and microprocessor, bandwidth improves by square of latency improvement**
  - **In the time that bandwidth doubles, latency improves by no more than 1.2X to 1.4X**
- **Lag probably even larger in real systems, as bandwidth gains multiplied by replicated components**
  - **Multiple processors in a cluster or even in a chip**
  - **Multiple disks in a disk array**
  - **Multiple memory modules in a large memory**
  - **Simultaneous communication in switched LAN**
- **HW and SW developers should innovate assuming Latency Lags Bandwidth**
  - **If everything improves at the same rate, then nothing really changes**
  - **When rates vary, require real innovation**

System "Solutions" for L. lags B.

"If a problem has no solution, it may not be a problem, but a fact not to be solved, but to be coped with over time." – Shimon Peres ("Peres's Law")

- **Leveraging capacity to help latency**
  - **Caching**, e.g.
    - Half the area of large multiprocessor chips is for caches
    - File system devote a large portion of MM for file cache
    - Disk drives include caches of many MB to avoid accessing disk surface
  - **Replication**, e.g.
    - ISPs use multiple sites across country to reduce network latency
    - Storage systems that replicate data for dependability read closest copy
    - Processor replicate registers in clusters of functional unit to reduce L.
- **Leveraging bandwidth to help latency**
  - **Prediction** e.g.
    - Increased use of prediction (speculation) techniques to avoid stalling CPU
      - Branch Prediction: Processor predict early "branch taken" allowing processor, caches, memory and disk controller to pre-fetch instruction/data.
      - Value Prediction
- Leveraging Cheap Computation e.g. Recalculate rather than fetch

Ref.: D. Patterson "Latency Lags Bandwidth", CACM Oct. 2004.

Little's Law (Revisited)

- Concurrency = Latency * Bandwidth
- Or, Expressed Concurrency = Latency * Effective Throughput
  - Bandwidth
    - conventional memory bandwidth
    - #floating-point units
  - Latency
    - memory latency
    - functional unit latency
  - Concurrency:
    - bytes expressed to the memory subsystem
    - concurrent (parallel) memory operations
- For example, consider a CPU with 2 FPU's each with a 4-cycle latency. Little's law states that we must express 8-way ILP to fully utilize the machine.
- Expressed Concurrency should increase by about 2*1.4 about 3x to keep up with progression of Latency and Bandwidth
  - Doubling of core (according to Moore's Law) not sufficient for overall system performance doubling.

Slide source: Samuel Williams, LBNL.

Computer System Balance Principle

(H. T. Kung "Memory Requirements for Balanced Computer Architecture", 1986)

- Processing Element (PE) characterized by
  - Computational Bandwidth (B_C)
  - I/O Bandwidth (B_{IO})
  - C_{comp}: computation cost (number of comp. operations)
  - C_{io} : I/O cost
  - Size of Local Memory (M)
- **PE is balanced iff Comp. Time = IO Time, i.e.**

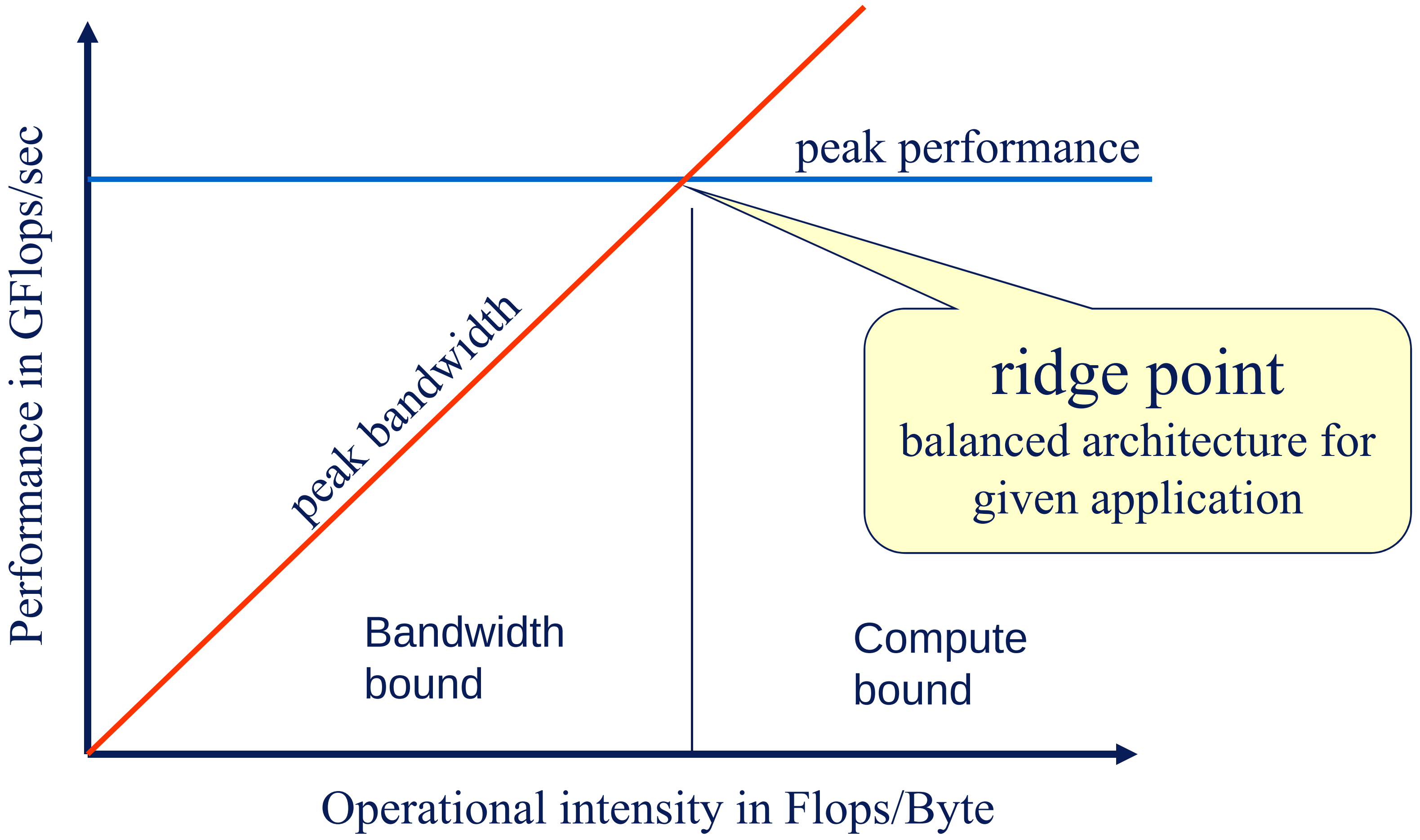
  $C_{comp}/B_C = C_{io}/B_{IO}$. Why?
  - **if B_C is increased** then PE will have to wait for I/O
  - **if B_C is decreased** then the system is overprovisioned (wasteful in cost)
- Role of LM: reduce overall I/O (alleviating I/O bottleneck)
- Balance Principle can be used to predict the growth in M needed to correct imbalance

Balance Principle Example

- According to B.P
  - $C_{comp}/B_C = C_{io}/B_{IO}$
  - i.e. $C_{comp}/C_{io} = B_C/B_{IO}$
- Suppose B_C is increased by α times B_{IO}
- To get C_{comp}/C_{io} we need a specific computation, say Matrix Multiplication
  - For Blocked MM (with block size $M^{1/2}$ x $M^{1/2}$)
    - $C_{comp}/C_{io} = \Theta(M^{1/2})$
    - $\Rightarrow M_{new} = \alpha^2 M$
- Since B_{IO} lacks behind B_C => Caches should become larger and larger to keep the architecture balanced.

Roofline Model

Introduced by Samual Williams and David Patterson (CACM April 2009).



peak performance
peak bandwidth
ridge point
balanced architecture for given application
Performance in GFlops/sec
Bandwidth bound
Compute bound
Operational intensity in Flops/Byte

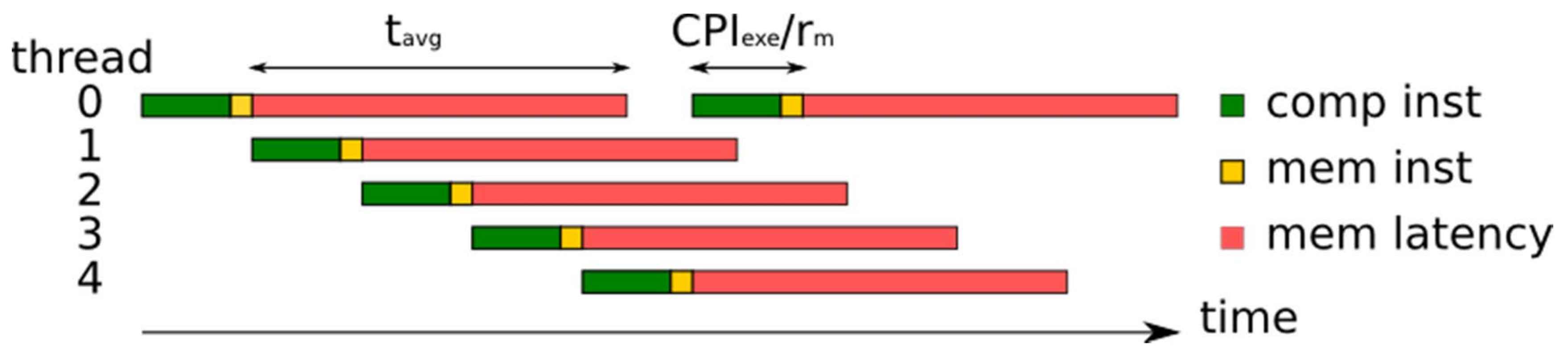
Slide: Henk Corporaal

Roofline Model

If memory latency can be hidden – app is either bandwidth bound or compute bound. Otherwise the program can be latency-bound.

r_m : percentage of memory instruction in total instruction
t_{avg} : average memory latency
CPI_{exe} : Cycle per Instruction

- There is one memory instruction in every ($1/r_m$) instructions.
- There is one memory instruction every ($1/r_m$) x CPI_{exe} cycles.
- It takes (t_{avg} x rm / CPI_{exe}) threads to hide memory latency.

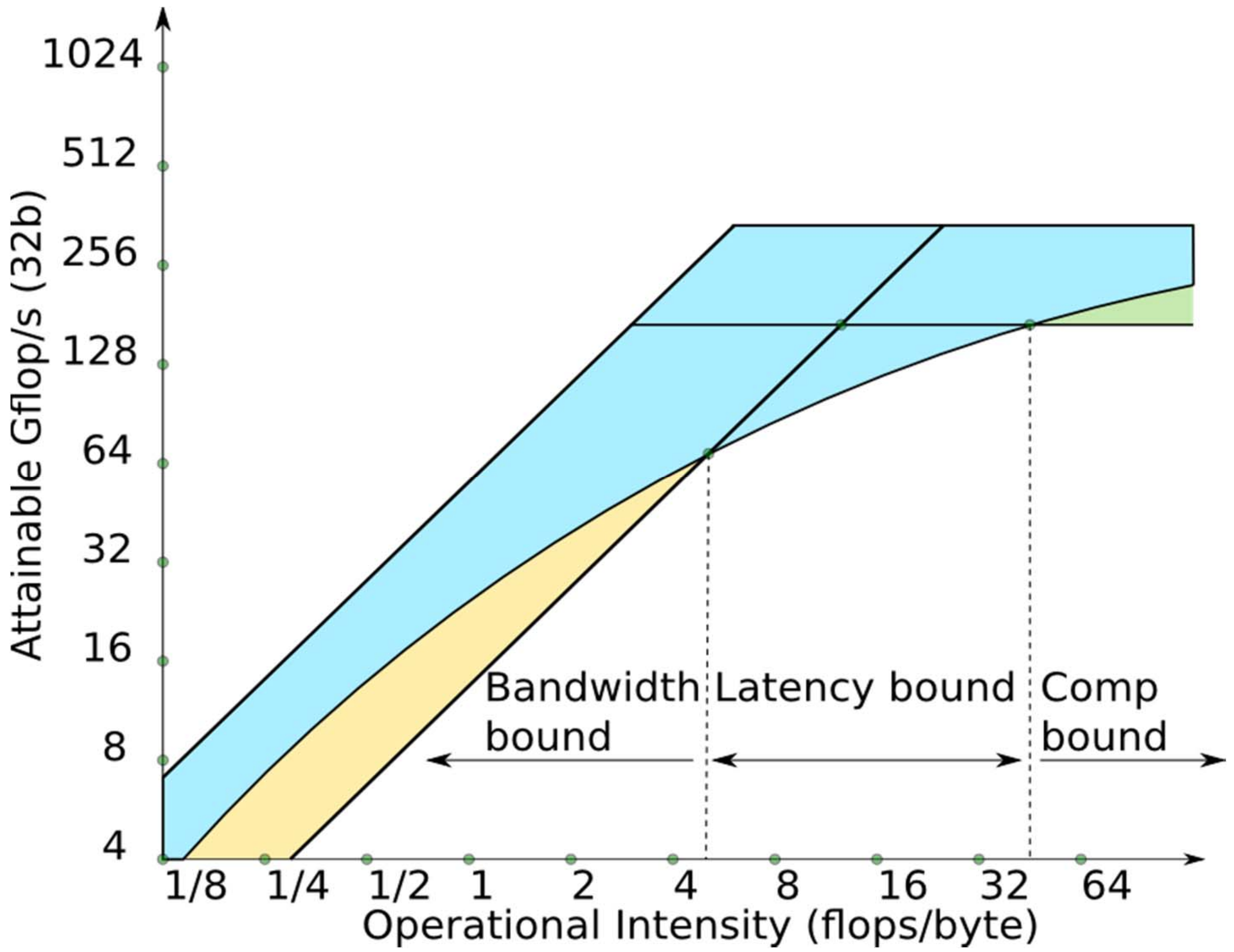


Z. Guz, et al, "Many-Core vs. Many-Thread Machines: Stay Away From the Valley", IEEE Comp Arch Letters, 2009, link
S. Hong, et al. "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness", ISCA09, link

Slide: Henk Corporaal

Roofline Model

If not enough threads to hide the memory latency, the memory latency could become the bottleneck.

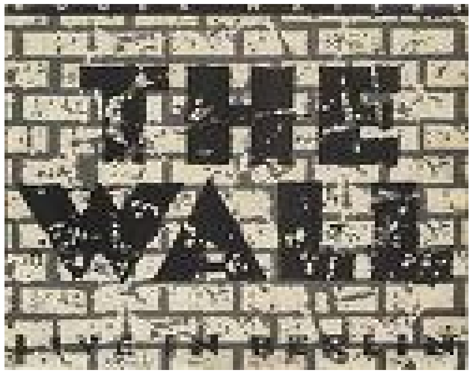


Samuel Williams, "Auto-tuning Performance on Multicore Computers", PhD Thesis, UC Berkeley, 2008, link
S. Hong, et al. "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness", ISCA09, link

Slide: Henk Corporaal

Why many core?

- Running into
  - Frequency wall
  - ILP wall
  - Memory wall
  - Energy wall

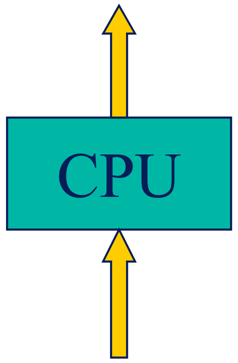


- Chip area enabler: Moore's law goes well below 22 nm
  - What to do with all this area?
  - Multiple processors fit easily on a single die
- Application demands
- Cost effective (just connect existing processors or processor cores)
- Low power: parallelism may allow lowering Vdd
  - Performance/Watt is the new metric !!

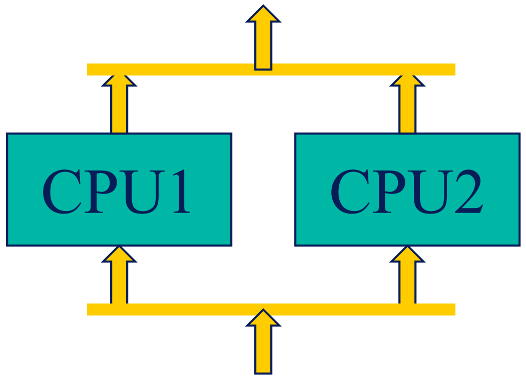
Slide: Henk Corporaal

Low power through parallelism

- Sequential Processor
  - Switching capacitance C
  - Frequency f
  - Voltage V
  - $P1 = \alpha f C V^2$



- Parallel Processor (two times the number of units)
  - Switching capacitance 2C
  - Frequency f/2
  - Voltage V’ < V
  - $P2 = \alpha f/2\ 2C\ V'^2 = \alpha f C V'^2 < P1$



Slide: Henk Corporaal

Designing Multicore Chips Hard

- Designers must confront single-core design options
  - Instruction fetch, wakeup, select
  - Execution unit configuation & operand bypass
  - Load/queue(s) & data cache
  - Checkpoint, log, runahead, commit.
- As well as additional design degrees of freedom
  - How many cores? How big each?
  - Shared caches: levels? How many banks?
  - Memory interface: How many banks?
  - On-chip interconnect: bus, switched, ordered?

Want Simple Multicore Hardware Model

To Complement Amdahl's Simple Software Model

(1) Chip Hardware Roughly Partitioned into

- Multiple Cores (with L1 caches)
- The Rest (L2/L3 cache banks, interconnect, pads, etc.)
- Changing Core Size/Number does NOT change The Rest

(2) Resources for Multiple Cores Bounded

- Bound of N resources per chip for cores
- Due to area, power, cost ($$$), or multiple factors
- Bound = Power? (but our pictures use Area)

Want Simple Multicore Hardware Model, cont.

(3) Micro-architects can improve single-core performance using more of the bounded resource

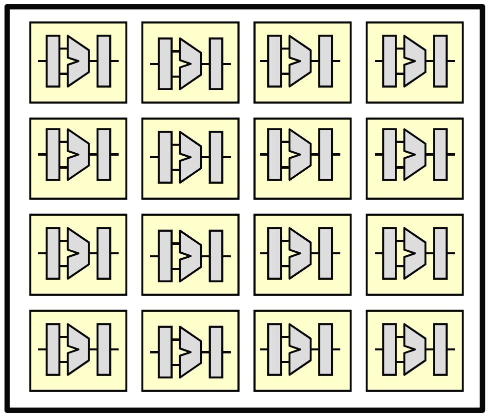
- A Simple Base Core
  - Consumes 1 Base Core Equivalent (BCE) resources
  - Provides performance normalized to 1
- An Enhanced Core (in same process generation)
  - Consumes R BCEs
  - Performance as a function Perf(R)
- What does function Perf(R) look like?

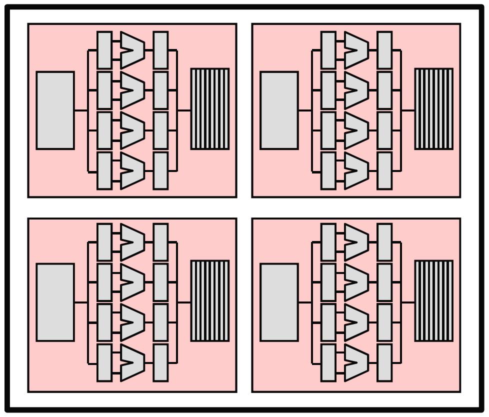
More on Enhanced Cores

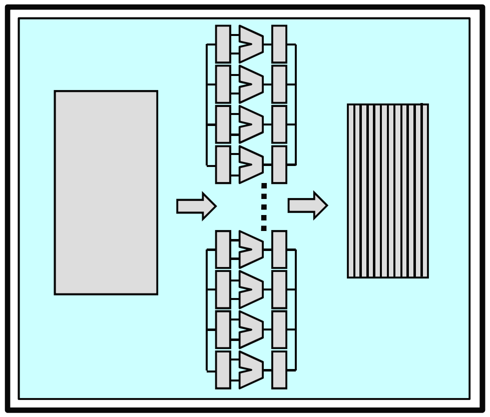
- (Performance Perf(R) consuming R BCEs resources)
- If Perf(R) > R ➔ Always enhance core
- Cost-effectively speedups both sequential & parallel
- Therefore, Equations Assume Perf(R) < R
- Graphs Assume Perf(R) = Square Root of R
  - 2x performance for 4 BCEs, 3x for 9 BCEs, etc.
  - Why? Models diminishing returns with "no coefficients"
  - Alpha EV4/5/6 [Kumar 11/2005] & Intel's Pollack's Law
- How to speedup enhanced core?
  - <Insert favorite or TBD micro-architectural ideas here>

Symmetric Multicore Model

- Each Chip Bounded to N BCEs (for all cores)
- Each Core consumes R BCEs
- Assume Symmetric Multicore = All Cores Identical
- Therefore, N/R Cores per Chip — (N/R)*R = N
- For an N = 16 BCE Chip:


Sixteen 1-BCE cores


Four 4-BCE cores


One 16-BCE core

Performance of Symmetric Multicore Chips

- Serial Fraction 1-F uses 1 core at rate Perf(R)
- Serial time = (1 – F) / Perf(R)

- Parallel Fraction uses N/R cores at rate Perf(R) each
- Parallel time = F / (Perf(R) * (N/R)) = F*R / Perf(R)*N

- Therefore, w.r.t. one base core:

$$\text{Symmetric Speedup} = \frac{1}{\underbrace{\dfrac{1 - F}{\text{Perf}(R)}} + \underbrace{\dfrac{F * R}{\text{Perf}(R)*N}}}$$

Enhanced Cores speed Serial & Parallel

- Implications?

Asymmetric (Heterogeneous) Multicore Chips

- Symmetric Multicore Required All Cores Equal
- Why Not Enhance Some (But Not All) Cores?

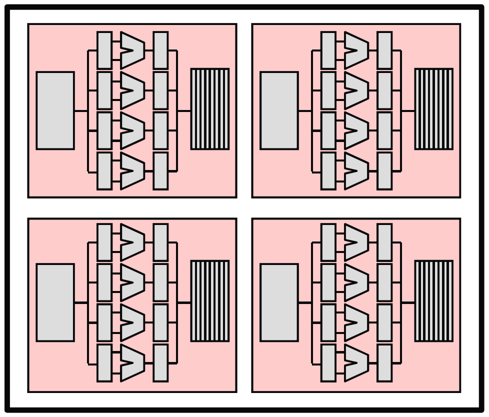
- For Amdahl's Simple Software Assumptions
  - One Enhanced Core
  - Others are Base Cores

- How?
  - <fill in favorite micro-architecture techniques here>
  - Model ignores design cost of asymmetric design

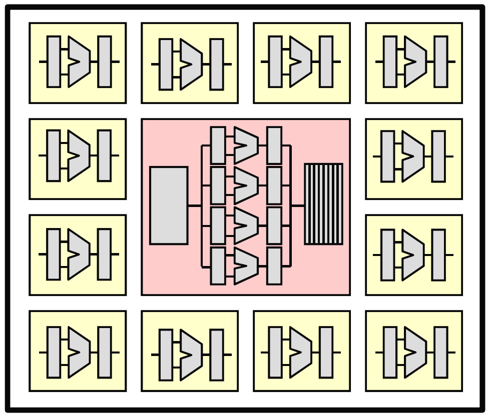
- How does this effect our hardware model?

Asymmetric Multicore Model

- Each Chip Bounded to N BCEs (for all cores)
- One R-BCE Core leaves N-R BCEs
- Use N-R BCEs for N-R Base Cores
- Therefore, 1 + N - R Cores per Chip
- For an N = 16 BCE Chip:



Symmetric: Four 4-BCE cores



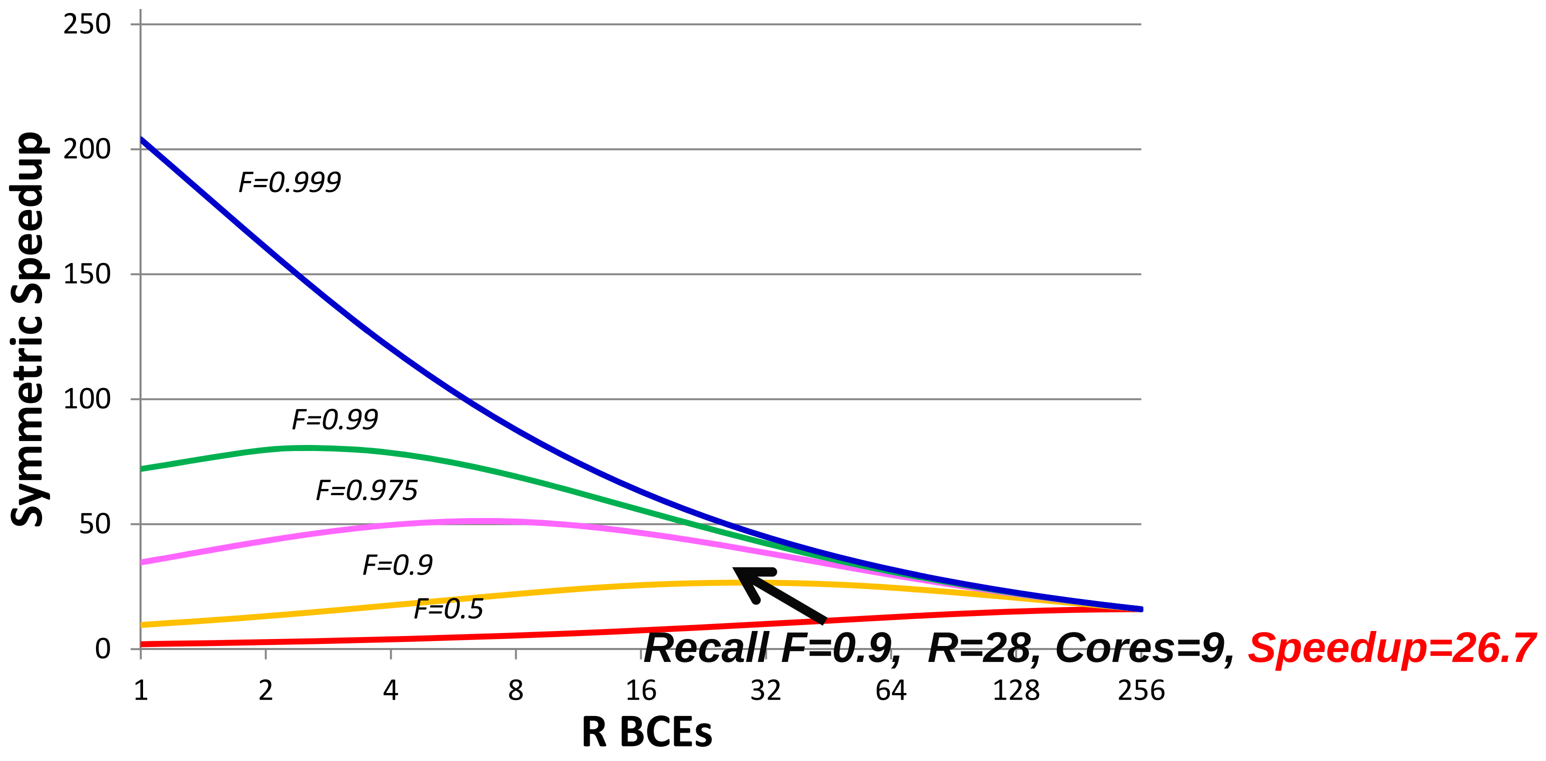
Asymmetric: One 4-BCE core & Twelve 1-BCE base cores

Performance of Asymmetric Multicore Chips

- Serial Fraction 1-F same, so time = (1 – F) / Perf(R)
- Parallel Fraction F
  - One core at rate Perf(R)
  - N-R cores at rate 1
  - Parallel time = F / (Perf(R) + N - R)
- Therefore, w.r.t. one base core:

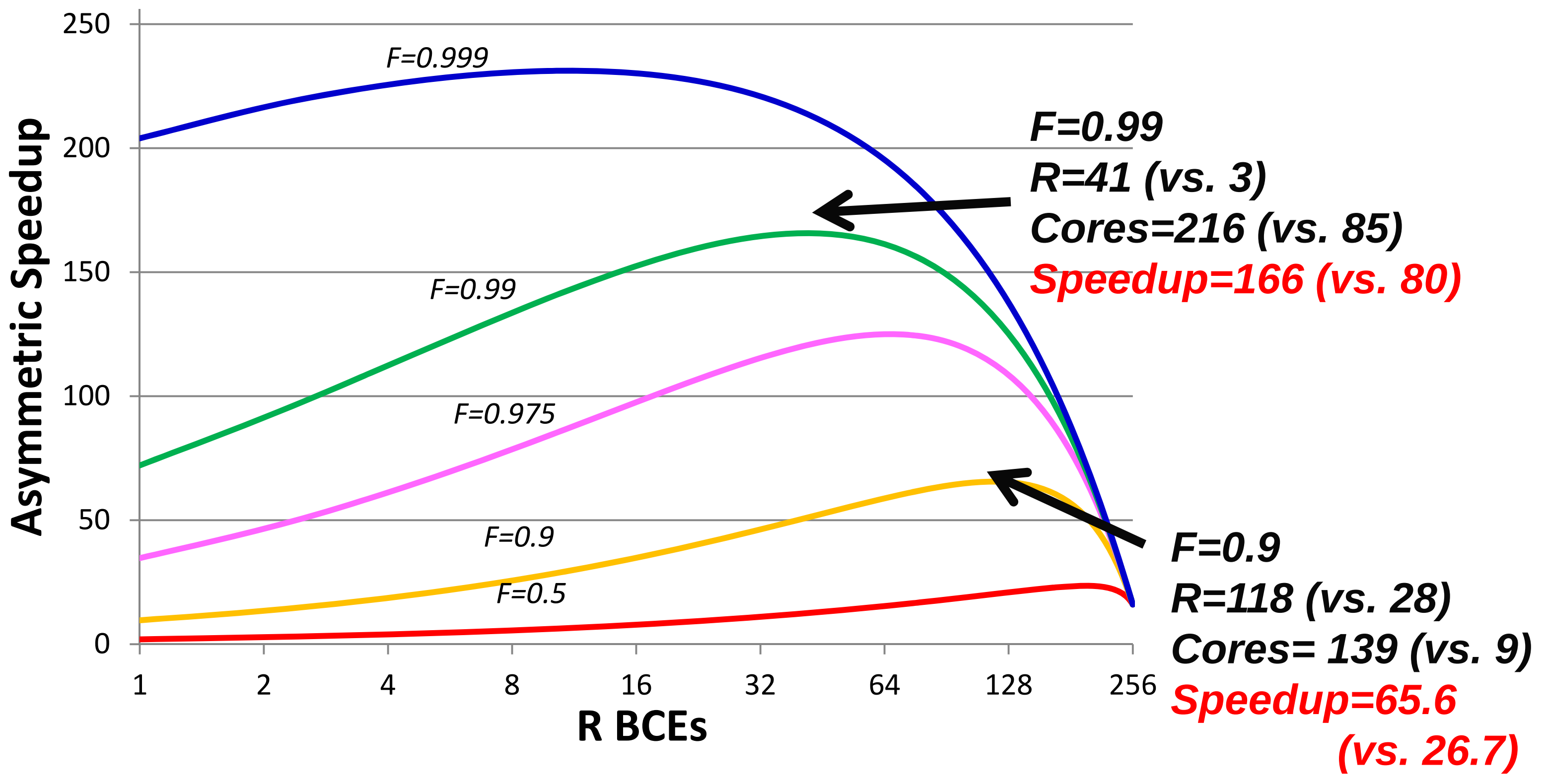
$$\text{Asymmetric Speedup} = \frac{1}{\frac{1 - F}{\text{Perf}(R)} + \frac{F}{\text{Perf}(R) + N - R}}$$

Recall Symmetric Multicore Chip, N = 256 BCEs



Recall F=0.9, R=28, Cores=9, Speedup=26.7

Asymmetric Multicore Chip, N = 256 BCEs



Asymmetric offers greater speedups potential than Symmetric

In Paper: As Moore's Law increases N, Asymmetric gets better

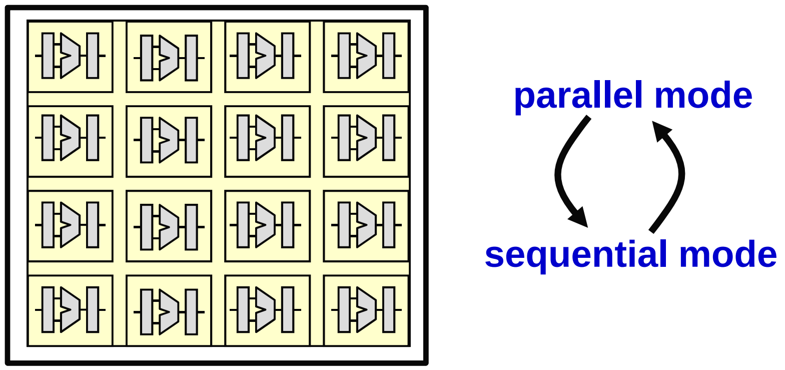
Some arch. researchers should target asymmetric multicores

Dynamic Multicore Chips

- Why NOT Have Your Cake and Eat It Too?

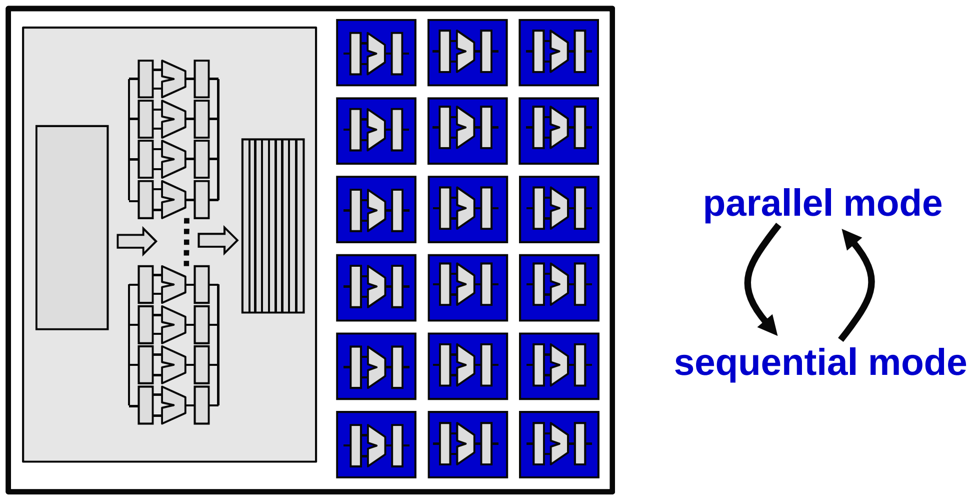
- N Base Cores for Best Parallel Performance
- Harness R Cores Together for Serial Performance

- How? DYNAMICALLY Harness Cores Together
  - <insert favorite or TBD techniques here>



Dynamic Multicore Chips, Take 2

- Let POWER provide the limit of N BCEs
- While Area is Unconstrained



How to model these two chips?

- Result: N base cores for parallel; large core for serial

Performance of Dynamic Multicore Chips

- N Base Cores with R BCEs used Serially

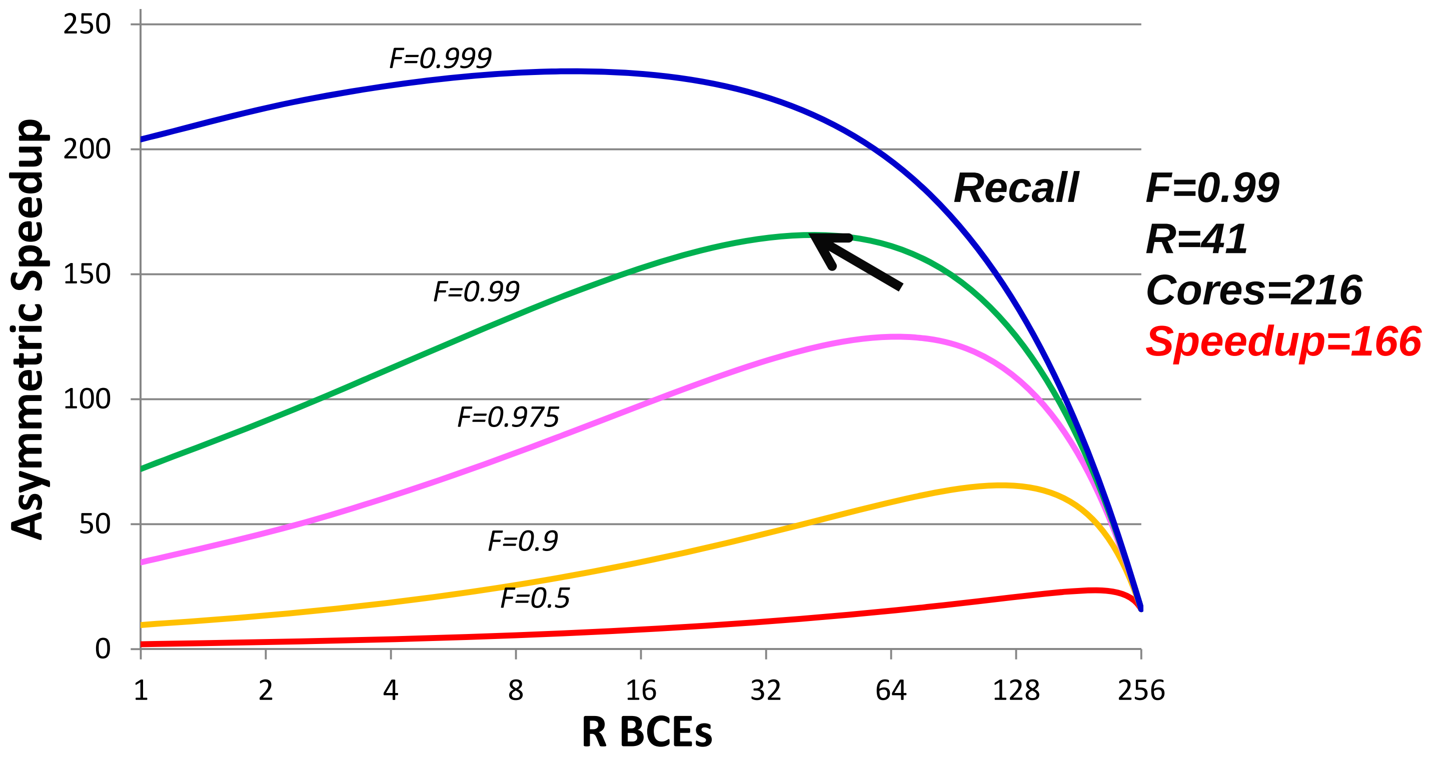
- Serial Fraction 1-F uses R BCEs at rate Perf(R)
- Serial time = (1 – F) / Perf(R)

- Parallel Fraction F uses N base cores at rate 1 each
- Parallel time = F / N

- Therefore, w.r.t. one base core:

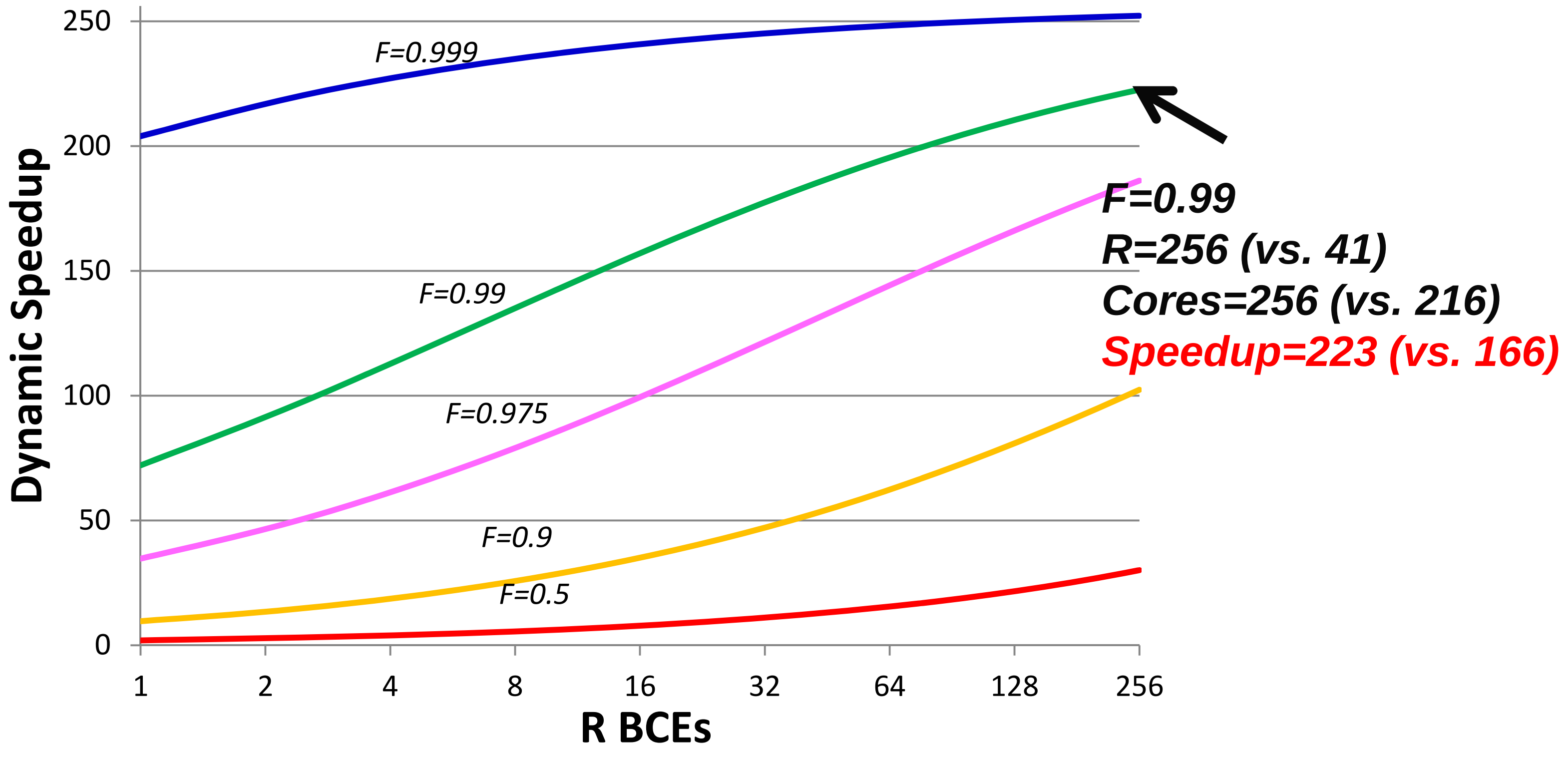
$$\text{Dynamic Speedup} = \frac{1}{\frac{1 - F}{\text{Perf}(R)} + \frac{F}{N}}$$

Recall Asymmetric Multicore Chip, N = 256 BCEs



What happens with a dynamic chip?

Dynamic Multicore Chip, N = 256 BCEs



Dynamic offers greater speedup potential than Asymmetric

Arch. researchers should target dynamically harnessing cores

Asym. & Dyn. Multicore: 3 Software Issues

1. Schedule computation (e.g., when to use bigger core)
2. Manage locality (e.g., sending code or data can sap gains)
3. Synchronize (e.g., asymmetric cores reaching a barrier)

At What Level?

- Application Programmer
- Library Author
- Compiler
- Runtime System
- Operating System
- Hypervisor (Virtual Machine Monitor)
- Hardware

Dynamic Challenges > Asymmetric Ones

Dynamic chips due to power likely

Three Multicore Amdahl's Law

Parallel Section

$$\text{Symmetric Speedup} = \cfrac{1}{\boxed{\cfrac{1-F}{Perf(R)}} + \cfrac{F * R}{\boxed{Perf(R)*N}}}$$

N/R Enhanced Cores

Sequential Section

1 Enhanced Core

$$\text{Asymmetric Speedup} = \cfrac{1}{\boxed{\cfrac{1-F}{Perf(R)}} + \cfrac{F}{\boxed{Perf(R) + N - R}}}$$

1 Enhanced & N-R Base Cores

$$\text{Dynamic Speedup} = \cfrac{1}{\boxed{\cfrac{1-F}{Perf(R)}} + \cfrac{F}{\boxed{N}}}$$

N Base Cores

Summary

- Simple Model of Multicore Hardware
  - Complements Amdahl's software model
  - Fixed chip resources for cores
  - Core performance improves sub-linearly with resources
- Research Implications

  (1) Need Dramatic Increases in Parallelism (No Surprise)
  - 99% parallel limits 256 cores to speedup 72
  - New Moore's Law: Double Parallelism Every Two Years?

  (2) Many larger chips need increased core performance

  (3) HW/SW for asymmetric designs (one/few cores enhanced)

  (4) HW/SW for dynamic designs (serial $\leftarrow\rightarrow$ parallel)

Amdahl's Law for Energy-Efficient Computing in Many-Core Era

- Two main factors limit growth of single chip:
  - Power supply (budget):
    - proportional to the energy cost for sustaining machines in data centers
    - Proportional to battery life of portable devices
  - Power density:
    - Pertains to thermal profile of the chip
      - Extra complexity (cooling fins etc) and cost

Ref: Woo & Lee "Extending Amdhal's Law for Energy-Efficient Computing in the Many Core Era, IEEE Computer 2008.

Many-core Models

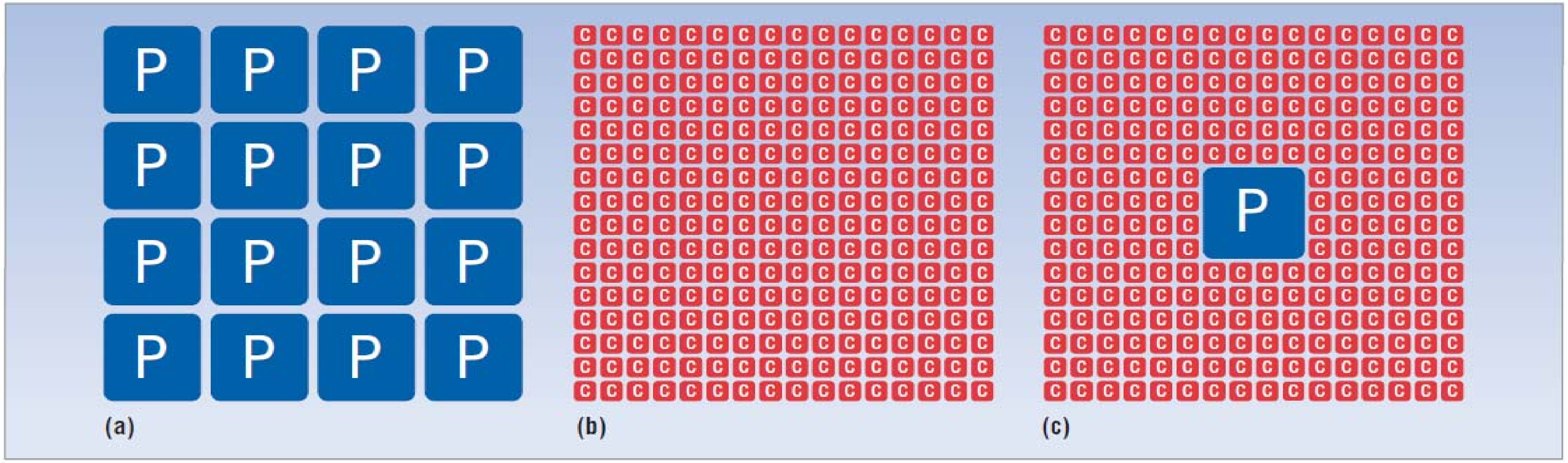


Figure 1. Many-core design styles. (a) A symmetric many-core processor that replicates a state-of-the-art superscalar processor on a die, and (b) a symmetric many-core processor that replicates a smaller, more power-efficient core on a die. (c) An asymmetric many-core processor with numerous efficient cores and one full-blown processor as the host processor.

- Many-core models
  - P* - Symmetric many-core with superscalar processors (fig (a))
  - c* - Symmetric many-core with smaller power efficient processors (fig (b))
  - P+c* - Asymmetric many core (fig ©)

Model Assumptions, Parameters, & Metrics

- A single P* core consumes unit power and has unit performacne
- k ($0 \leq k \leq 1$): frac. of power P* core consumes when **idle**
- s_c ($0 \leq s_c \leq 1$): small (c) core's perf. w.r.t. P* core's
- w_c ($0 \leq w_c \leq 1$): c core's power consumption w.r.t. P* core's
- k_c ($0 \leq k_c \leq 1$):frac. of power c core consumes when **idle w.r.t. to its own power consumption**
- **Perf: performance (= 1/Execution Time)**
- **W: avg. power consumption**
- **Perf/W : performance per watt –** performance achievable at the same cooling capacity, based on the average power W.
  - **= 1/Energy Consumption since Perf = 1/Execution Time**
  - **Note: Perf/W for single core is 1.**
- **Perf/J: performance per Joule (= 1/energy*delay)**
  - **Performance for same battery life (useful for portable devices)**
- **$s_c = \sqrt{w_c}$ (due to Pollack's rule Perf = $\sqrt{}$ Area and Power α Area)**

Power Equivalent Modeling

- W_{budget} : Power budget (of a single-chip)
  - Let P* core power = 20W, c core power = 5W, and W_{budget} of 160W
    - Only 8 P* cores whereas 32 c core chip
- Power-limited equivalent design
  - n_{P*} : maximum P* core which can be implemented within W_{budget}
  - $n_{c*} = W_{budget} / w_c$ – num. c cores within W_{budget}
  - nP+c* = $1+(W_{budget} -1)/w_c$ – num of cores on P+c* chip

Model P*

- According to Amdahl's law $Perf = \frac{1}{(1-f)+\frac{f}{n}}$

- Power during seq. phase = 1 + (n-1)k; parallel phase = n
- Average Power W: $\longrightarrow$

$$W = \frac{(1-f)\times\{1+(n-1)k\}+\frac{f}{n}\times n}{(1-f)+\frac{f}{n}}$$
$$= \frac{1+(n-1)k(1-f)}{(1-f)+\frac{f}{n}}$$

- Perf/W:

Perf/J = Perf * Perf/W:

$$\frac{Perf}{W} = \frac{1}{(1-f)+\frac{f}{n}} \times \frac{(1-f)+\frac{f}{n}}{1+(n-1)k(1-f)}$$
$$= \frac{1}{1+(n-1)k(1-f)}$$

$$\frac{Perf}{J} = \frac{1}{(1-f)+\frac{f}{n}} \times \frac{1}{1+(n-1)k(1-f)}$$

P* scalability

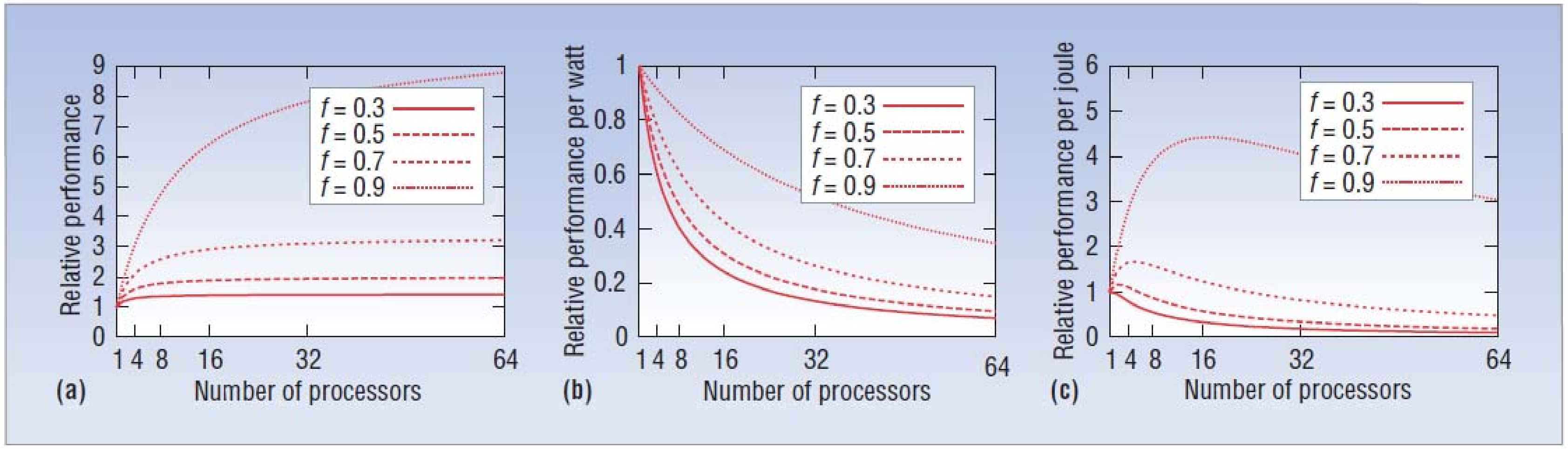


Figure 2. P scalability. P*, a symmetric many-core processor that replicates a state-of-the-art superscalar processor on a die, consumes a high amount of energy to complete the task: (a) performance, (b) performance per watt, and (c) performance per joule, where* k = *0.3.*

- Observations:
  - Perf/W seq = Perf/W parallel execution only for f = 1 (i.e. only when perf. improvement using parallelization improves linearly)
  - Otherwise Perf/W parallel < 1 and decreases with n.
  - Balancing load is important for both power-supply efficiency and extended battery life.

Model c*

- According to Amdahl's law $$Perf = \frac{s_c}{(1-f)+\frac{f}{n}}$$
- Power during seq. phase = $w_c + (n\text{-}1)\, w_c k_c$; parallel phase = nw_c
- Average Power W: $\longrightarrow$

$$W = \frac{\frac{1-f}{s_c}\times\{w_c+(n-1)w_c k_c\}+\frac{f}{ns_c}\times nw_c}{\frac{1-f}{s_c}+\frac{f}{ns_c}}$$

$$= \frac{w_c+(n-1)w_c k_c(1-f)}{(1-f)+\frac{f}{n}}$$

- Perf/W:

$$\frac{Perf}{W} = \frac{s_c}{w_c+(n-1)w_c k_c(1-f)}$$

Perf/J = Perf * Perf/W:

$$\frac{Perf}{J} = \frac{s_c}{(1-f)+\frac{f}{n}} \times \frac{s_c}{w_c+(n-1)w_c k_c(1-f)}$$

c* scalability

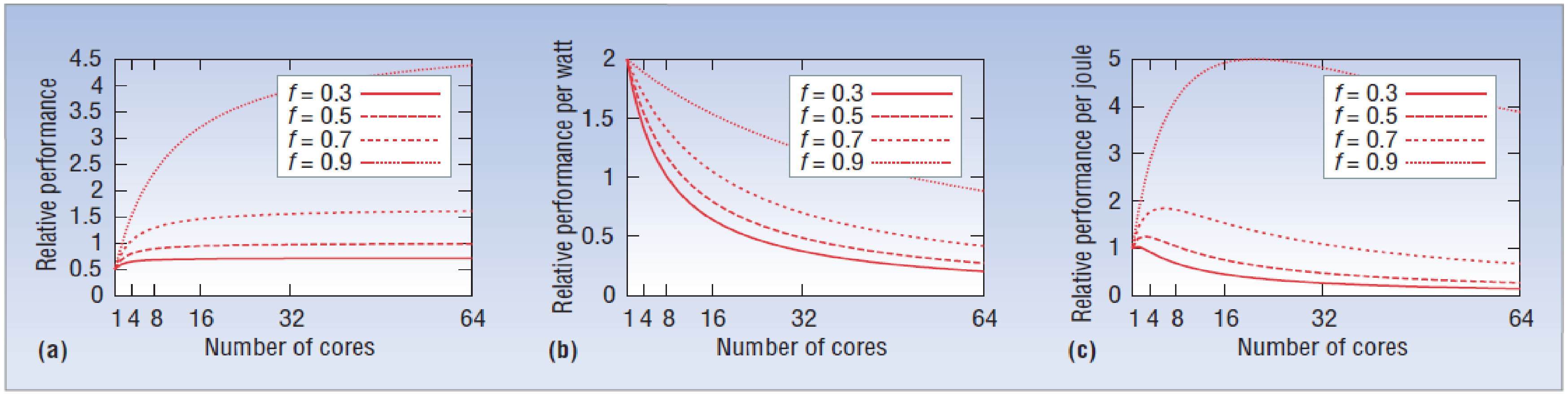


Figure 3. c scalability. The maximum speedup of this c*—a symmetric many-core processor that replicates a smaller, more power-efficient core on a die—isn't as high as that of P*: (a) performance, (b) performance per watt, and (c) performance per joule, where $s_c = 0.5$, $w_c = 0.25$, and $k_c = 0.2$.*

- Observations:
  - Max perf of c* is not as high as P* - limited by seq performance as per Amdahl's law.
  - Perf/W and Perf/J better than that for full blown processor only for small n.

Model P+c*

- According to Amdahl's law
$$Perf = \frac{1}{(1-f) + \frac{f}{(n-1)s_c}}$$

- P during seq. phase = 1 + (n-1) $w_c k_c$; parallel phase = k +(n-1)w_c
- Average Power W: $\longrightarrow$
$$W = \frac{(1-f)\{1+(n-1)w_c k_c\} + \frac{f}{s_c}\left\{\frac{k}{n-1} + w_c\right\}}{(1-f) + \frac{f}{(n-1)s_c}}$$

- Perf/W:

$$\frac{Perf}{W} = \frac{1}{(1-f)\{1+(n-1)w_c k_c\} + \frac{f}{s_c}\left\{\frac{k}{n-1} + w_c\right\}}$$

Perf/J = Perf * Perf/W:

$$\frac{Perf}{J} = \frac{1}{(1-f) + \frac{f}{(n-1)s_c}} \times \frac{1}{(1-f)\{1+(n-1)w_c k_c\} + \frac{f}{s_c}\left\{\frac{k}{n-1} + w_c\right\}}$$

P+c* scalability

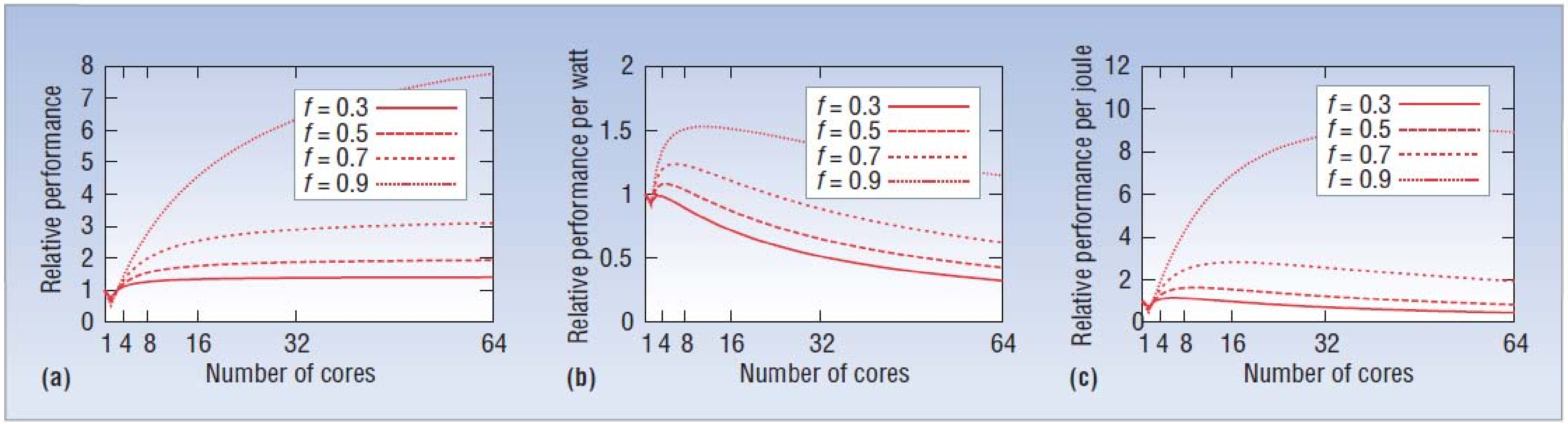


Figure 4. P + c scalability. P + c* is an asymmetric many-core processor with numerous efficient cores and one full-blown processor as the host processor: (a) performance, (b) performance per watt, and (c) performance per joule, where* k = *0.3,* $s_c = 0.5$, $w_c = 0.25$, *and* $l_c = 0.2$.

- Observations:
  - Unlike P* and c* whose Perf/W monotonically decreases, an optimal number of cores exists that consumes the least amount of energy for parallel execution.
  - However, Perf/W becomes worse than one-core baseline when the numebr of cores exceeds a certain peak
  - Perf/J is much better than P* and c* - suitable for embedded apps.

Power-Equivalent Comparison

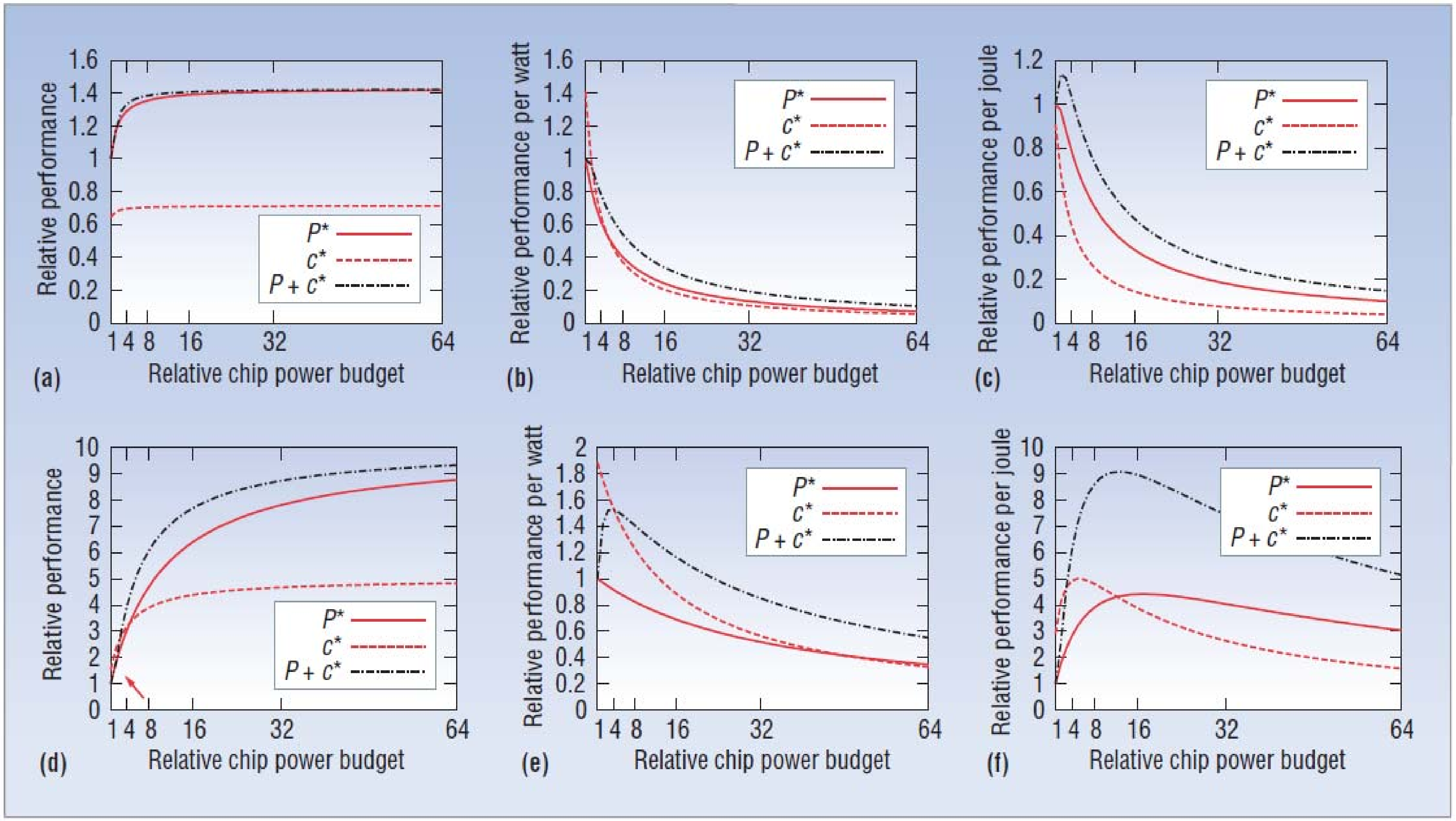


Figure 5. Power-equivalent models. We used power-equivalent models to perform cross-design comparisons. Given f = 0.3, we measured (a) performance, (b) performance per watt, and (c) performance per joule. Given f = 0.9, we measured (d) performance, (e) performance per watt, and (f) performance per joule.

C* is better for large f and small power budget

Summary

- ❑ Computer Architecture need to take into account technology trend.
- ❑ Amdahl’s Law and Little’s Law are important and applicable for multicore
- ❑ Principle of Balanced Systems and Roofline model help to maximize perf/cost.
- ❑ Simple models can provide deep insight into where to focus efforts.
- ❑ Perf/W and Perf./J are new metrics for evaluating computer systems.