# Građa računala

## Paralelizam II - Paralelizam i heterogenost (asimetričnost)

# Potencijalna dodatna literatura

- Grochowski et al., “Best of Both Latency and Throughput,” ICCD 2004.
- Suleman et al., “Accelerating Critical Section Execution with Asymmetric Multi-Core Architectures,” ASPLOS 2009, IEEE Micro Top Picks 2010.
- Joao et al., “Bottleneck Identification and Scheduling in Multithreaded Applications,” ASPLOS 2012.
- Joao et al., “Utility-Based Acceleration of Multithreaded Applications on Asymmetric CMPs,” ISCA 2013.
- Dakic et al., “Optimizing Kubernetes Performance, Efficiency and Energy Footprint in Heterogenous HPC Environments,” DAAAM 2021.

# Heterogenost (Asimetričnost) → Specijalizacija

- Pojmovi heterogenosti i asimetričnosti - isto značenje
- Heterogenost je generalni koncept dizajna sustava (i drugih stvari u životu)
- Ideja: umjesto više instanci istog resursa (homogeni, simetrični sustav), dizajnirajmo neke instance da budu drugačije (heterogene, asimetrične)
- Različite instance mogu biti optimizirane za veći efikasnost kod izvršavanja različitih tipova zadataka i/ili za zadovoljavanje drugačijih dizajn kriterija
  - to zapravo omogućuje specijalizaciju i prilagodbu

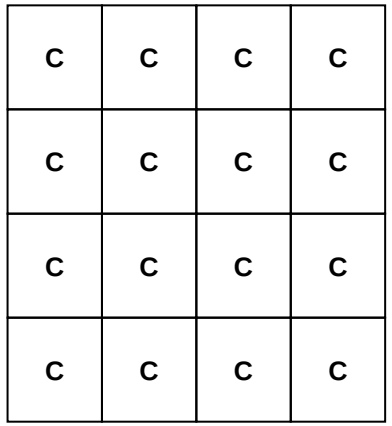
# Zašto asimetričnost - I

- Različiti workload-i se drugačije ponašaju
  - različite aplikacije - različito ponašanje
  - različite faze izvršavanja aplikacije - različito ponašanje
  - ista aplikacija koja se izvršava u različito vrijeme može se drugačije ponašati (zbog promjena na ulazu, dinamičkih uvjeta okoline)
  - Npr. lokalnost, bolja predvidivost grananja, paralelizma na razini instrukcija, međuovisnosti podataka, ...
- Sustavi su dizajnirani za različite metrike u isto vrijeme
  - Gotovo nikad nemamo samo jedan cilj u dizajnu
  - Npr. performanse, energetska efikasnost, pravednost, predvidivost, pouzdanost, dostupnost, trošak, kapacitet memorije, latencija, ...

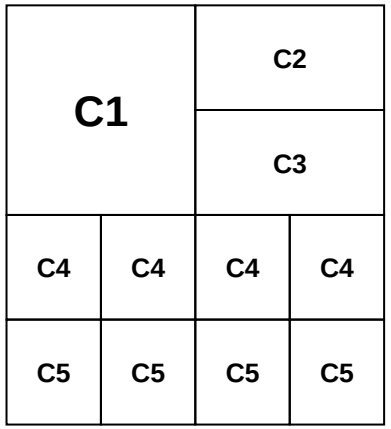
# Zašto asimetričnost - II

- Problem: Simetrični dizajn je one-size-fits-all
- Pokušava jedan dizajn prilagoditi svim metrikama i aplikacijama
- Vrlo je teško napraviti takav dizajn
  - koji zadovoljava sve aplikacije čak i za samo jednu metriku
  - koji zadovoljava sve dizajn metrike za sve aplikacije u isto vrijeme
- Ovo vrijedi i za komponente i resurse
  - cache memorija, jezgre, memorijski kontroleri, sabirnice, diskovi, serveri, ...
  - algoritmi, politike, ...

# Asimetričnost omogućava prilagodbu




Symmetric



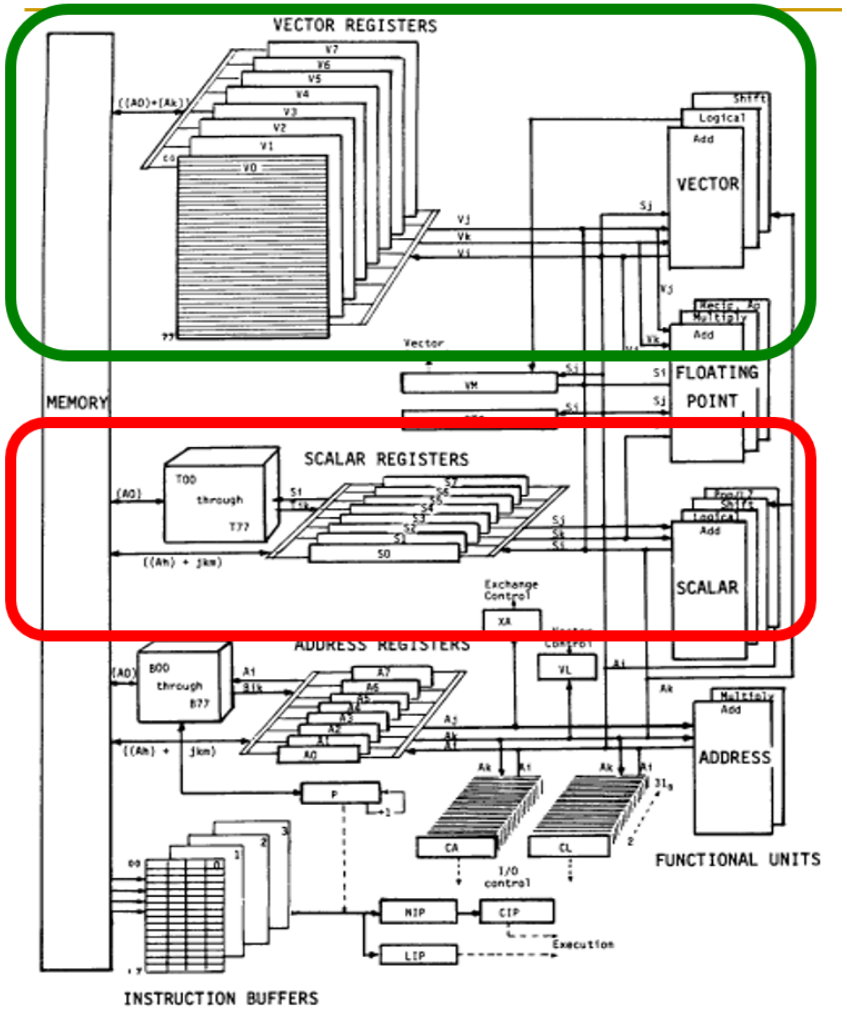

Asymmetric

- Simetrično: One size fits all
  - Potrošnja energije i performanse nisu optimalni za različite aplikacije
- Asimetrično: Prilagodba
  - Potreba za procesiranjem varira s obzirom na aplikaciju
  - Pokretanje koda na resursu koji je najbolji fit

# Primjeri, prije spomenuti

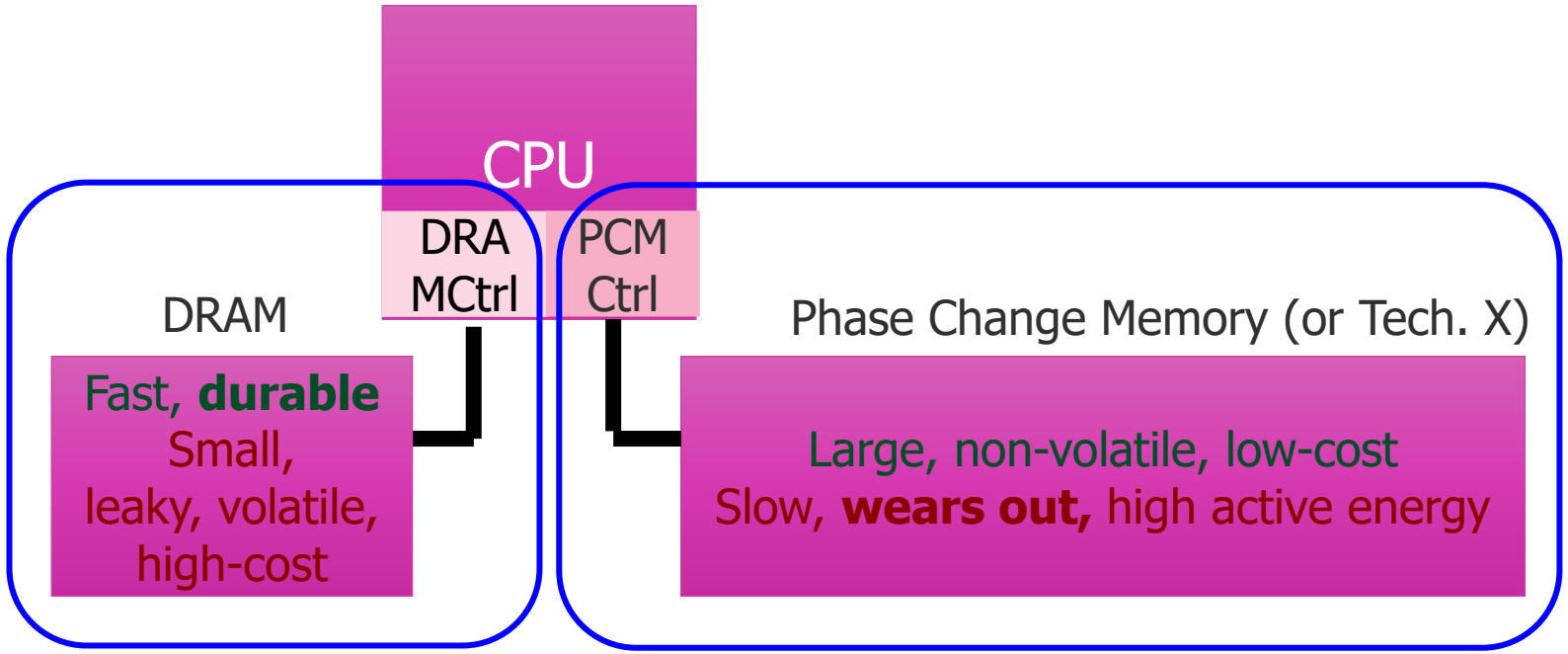
- CRAY-1 design: scalar + vector pipelines
- Modern processors: scalar instructions + SIMD extensions
- Decoupled Access Execute: access + execute processors
- Thread Cluster Memory Scheduling: different memory scheduling policies for different thread clusters
- RAIDR: Heterogeneous refresh rates in DRAM
- Heterogeneous-Latency DRAM (Tiered Latency DRAM)
- Hybrid memory systems
  - DRAM + Phase Change Memory
  - Fast, Costly DRAM + Slow, Cheap DRAM
  - Reliable, Costly DRAM + Unreliable, Cheap DRAM
- Heterogeneous cache replacement policies

# Primjer asimetričnog dizajna: CRAY-1




- CRAY-1
- Russell, "The CRAY-1 computer system," CACM 1978.
- skalarni i vektorski način rada
- 8 64-element vector registara
- 64 bits po elementu
- 16 memory banks
- 8 64-bit skalarnih registara
- 8 24-bitnih adresnih registara

# Prisjetimo se: hibridni memorijski sustavi




Hardware/software upravljaju alokacijom memorije kako bi se postiglo najbolje od više tehnologija odjednom

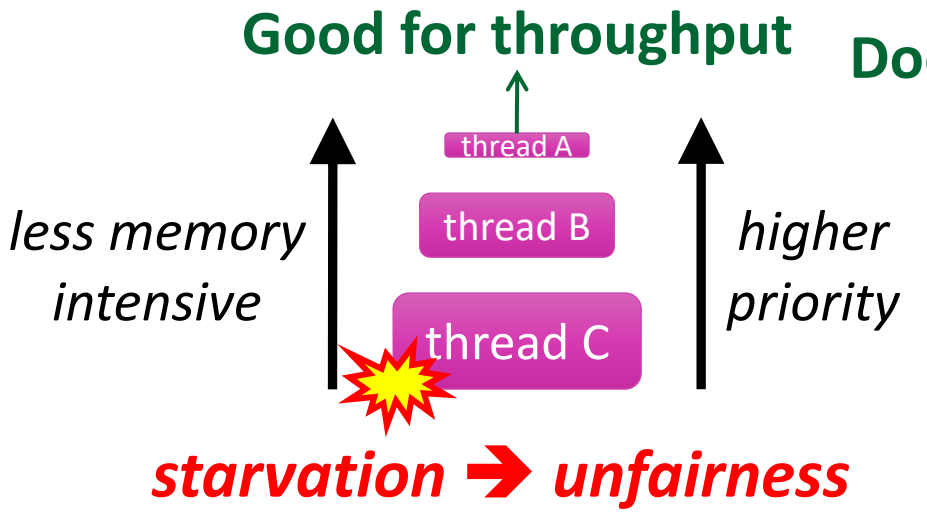
# Podsjetimo se: Propusnost vs. Pravednost

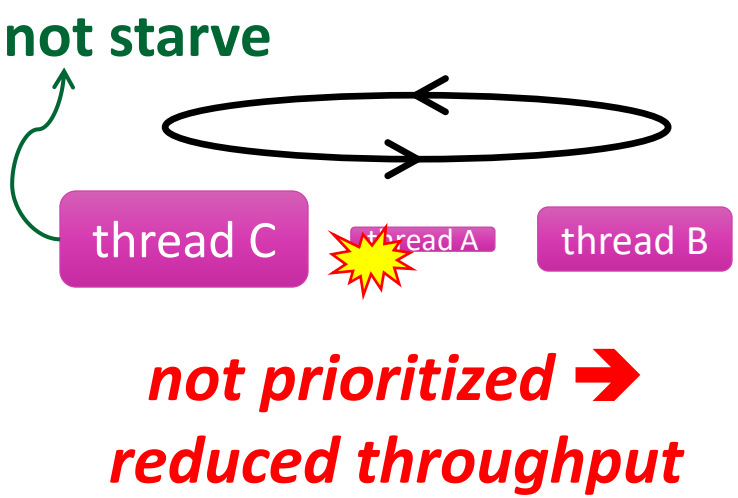
**_Throughput biased_** *approach*

Prioritizacija threadova koji su manje intenzivni memorijski

**_Fairness biased_** *approach*

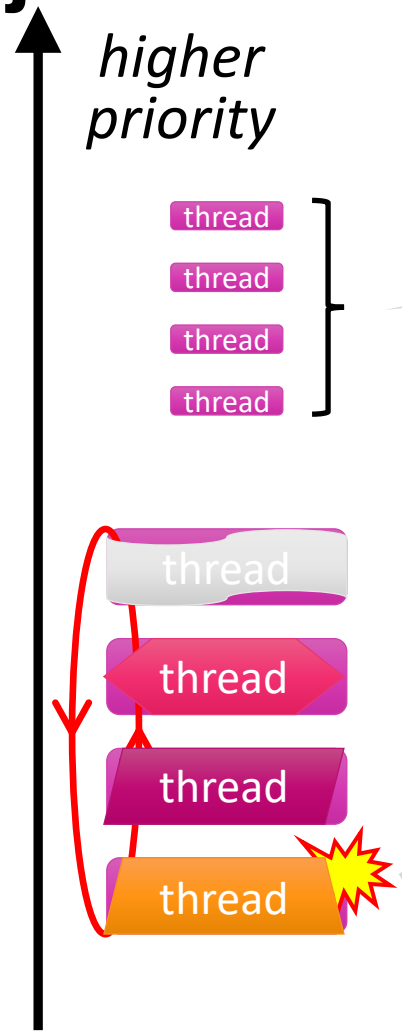
pristup memoriji "na parove razbroj s' "




**Single policy for all threads is insufficient**

# Podsjetimo se: Najbolje od oba svijeta




***For Throughput***

**Prioritize memory-non-intensive threads**

***For Fairness***

**Unfairness caused by memory-intensive being prioritized over each other**

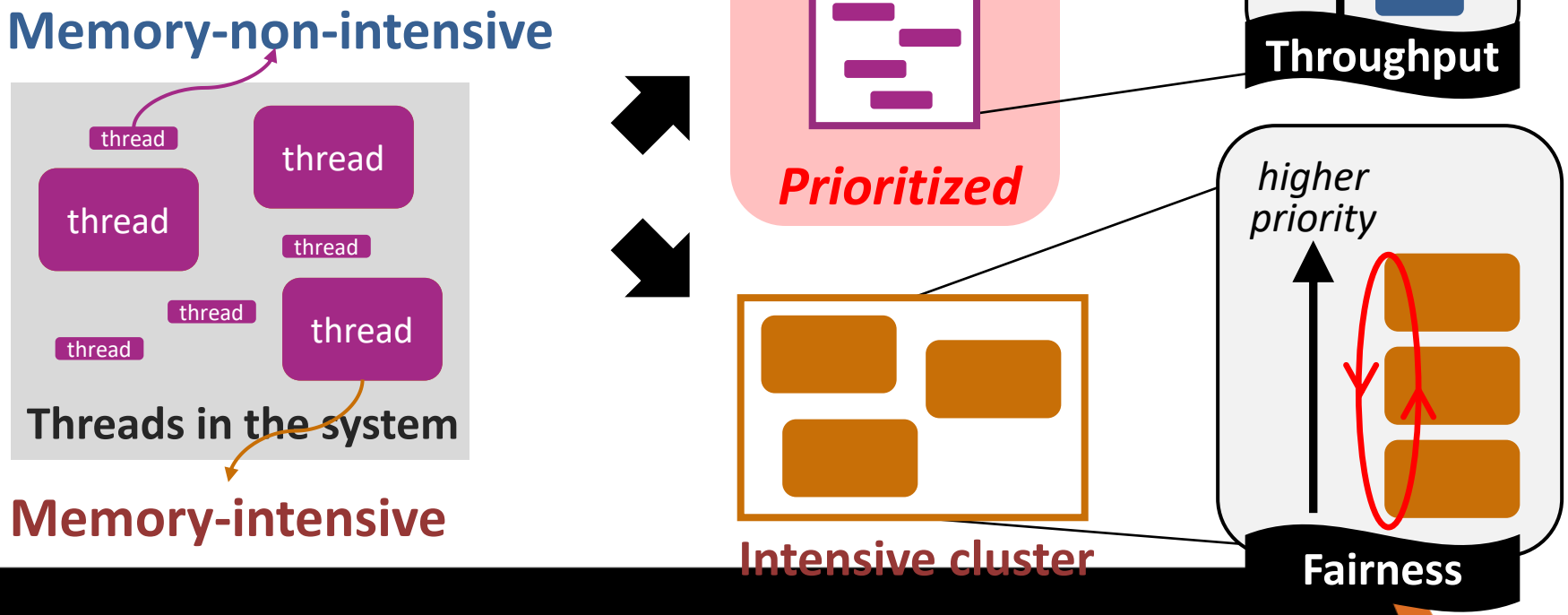
- Shuffle thread ranking

**Memory-intensive threads have different vulnerability to interference**

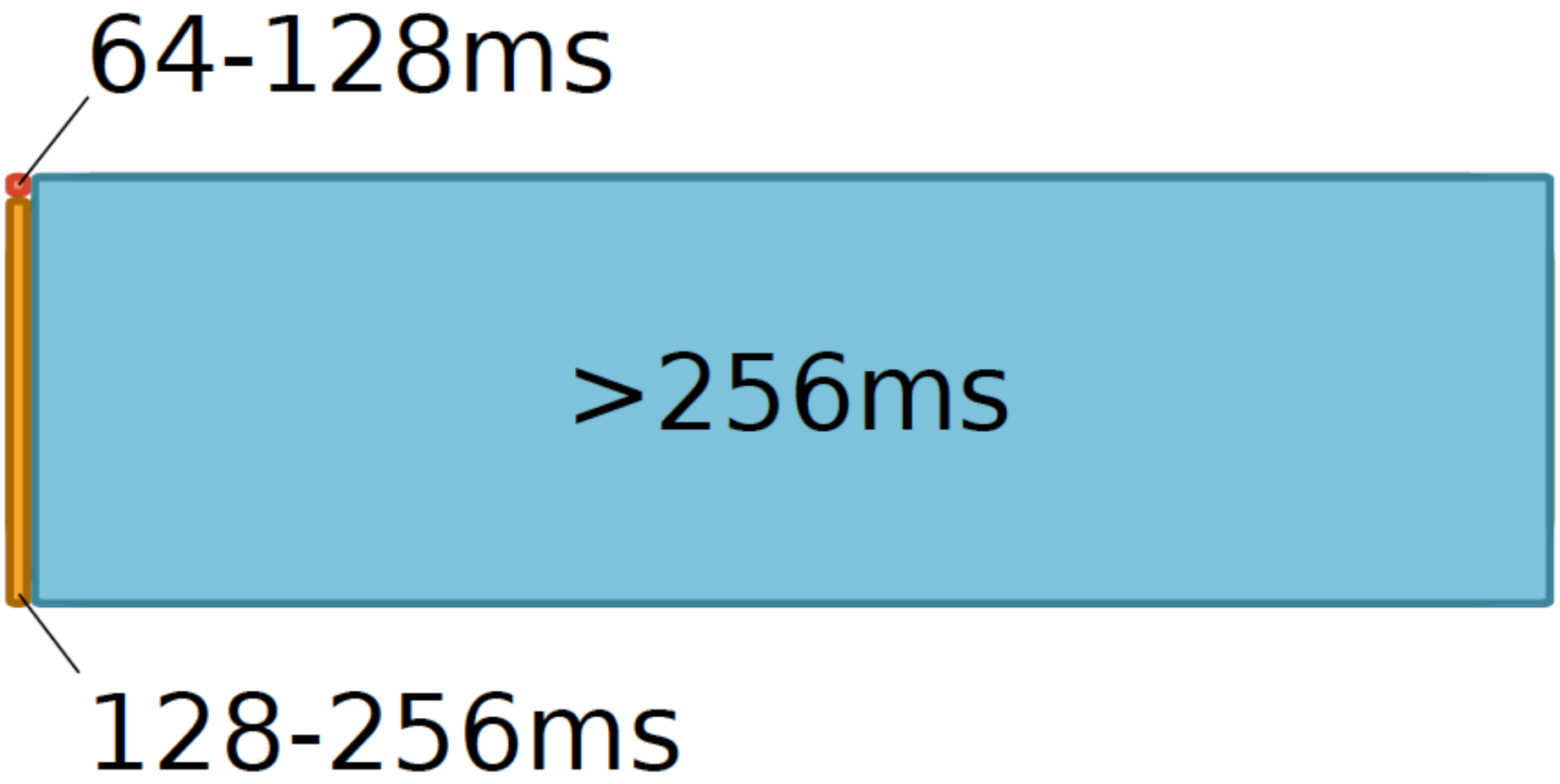
- Shuffle asymmetrically

# Scheduling threadova na razini memorije klastera

1. Grupiranje threadova u *clustere*
2. Prioritiziraj klaster koji nije pod velikim opterećenjem
3. Različite politike za svaki klaster



# Prisjetimo se: vrijeme zadržavanja podataka u RAM-u



# Heterogeno povezivanje (Tilera)

- 2D Mesh
- Pet mreža
- Prospajanje na razini četiri paketa
  - Dimension order routing, wormhole flow control
  - TDN: Cache request packets
  - MDN: Response packets
  - IDN: I/O packets
  - UDN: Core to core messaging
- Prospajanje na razini jednog sklopa
  - STN: Low-latency, high-bandwidth static network
  - Streaming data

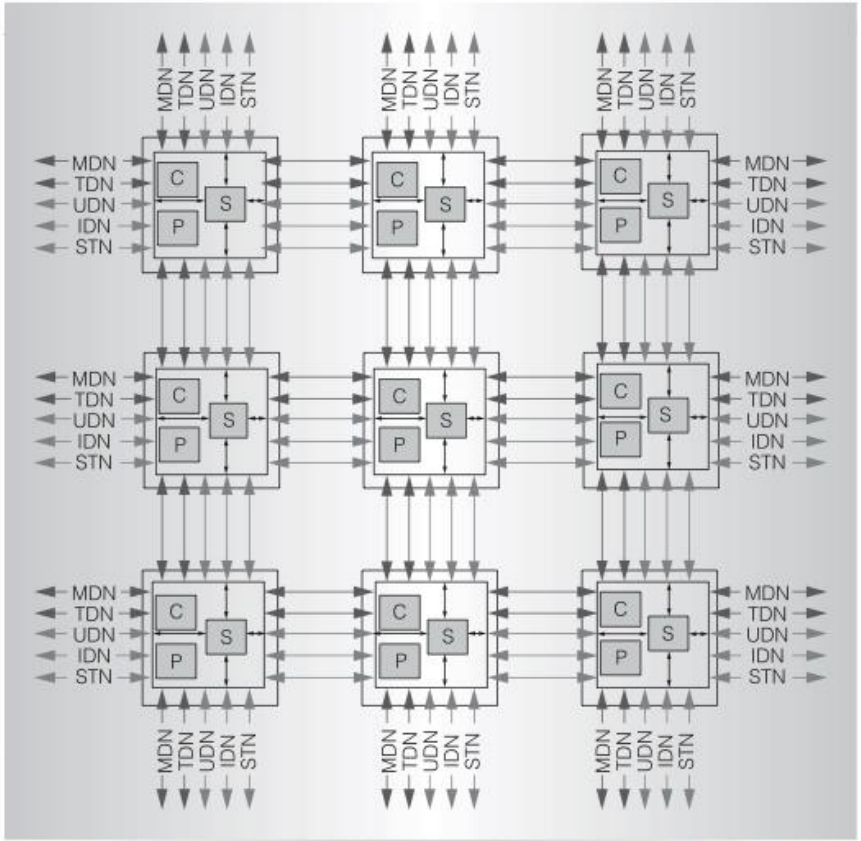



Figure 3. A 3 × 3 array of tiles connected by networks. (MDN: memory dynamic network; TDN: tile dynamic network; UDN: user dynamic network; IDN: I/O dynamic network; STN: static network.)

# Primjeri iz realnog života

- Heterogenost je svugdje oko nas
  - u prirodi i komponentama koje izrađujemo
- Ljudi su heterogeni
- Stanice su heterogene (specijaliziranost)
- Organi su heterogeni
- Auti su heterogeni
- Zgrade su heterogene
- ....

# General-Purpose vs. Special-Purpose

- Ideja asimetričnosti je - omogućiti specijalizaciju
- Da možemo premostiti jaz između čiste GP i SP arhitekture
  - čista GP: jedan dizajn za sve
  - čista SP: jedan dizajn po aplikaciji ili metrici
  - asimetrično: više pod-dizajna koji su optimizirani za neki set aplikacija i grupirani zajedno
- Cilj dobrog asimetričnog dizajna je da dobijemo najbolje od GP i SP svjetova

# Asimetričnost - prednosti i mane

- Prednosti
  + omogućava optimizaciju za više metrika
  + može omogućiti bolju prilagodbu aplikaciji
  + daje SP benefite sa GP fleksibilnošću

- Mane
  - veći overhead i kompleksnost dizajna i verifikacije
  - veći overhead u upravljanju (scheduling)
  - overhead u prebacivanju između više komponenti može dovesti do degradacije performansi

# Još jedan primjer

- Moderni procesori integriraju i jezgre za općenite zadatke i GPU jezgre
  - CPU-GPU sustavi
  - heterogenost u izvršavanju i ISA-i

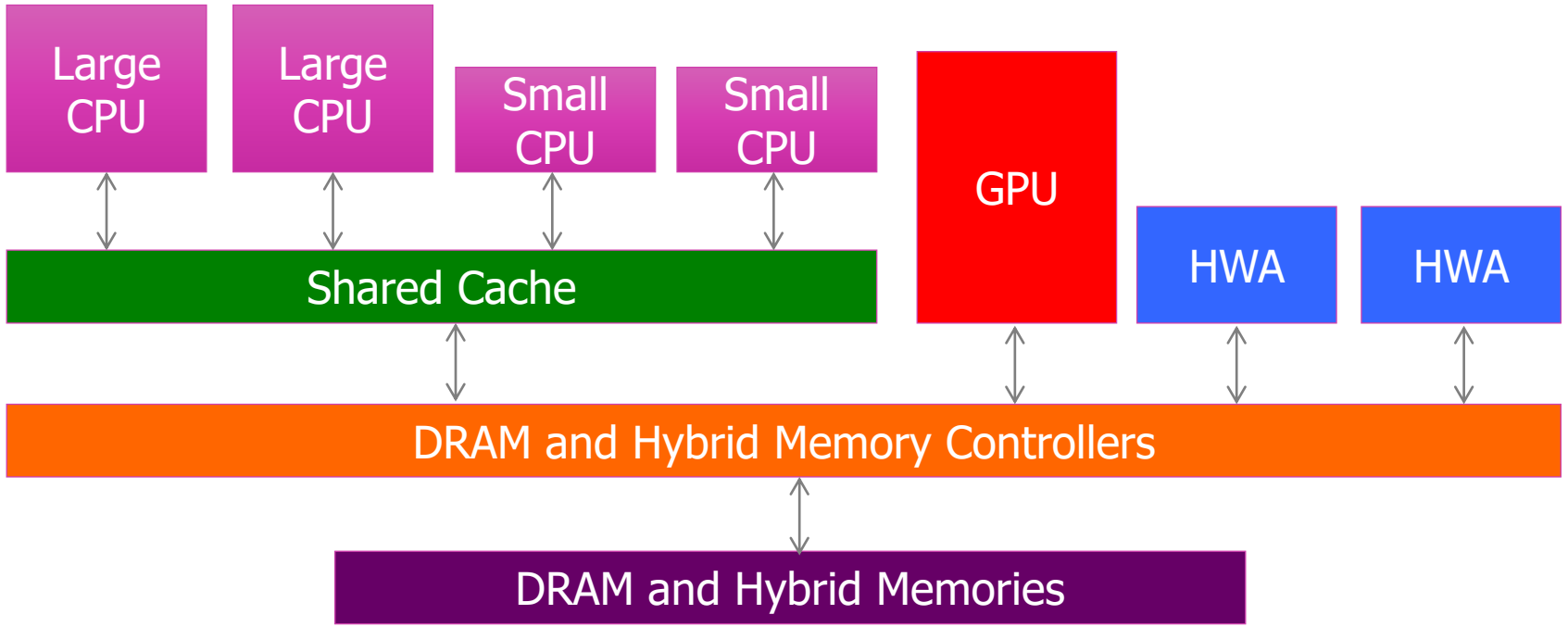
# Tri ključna problema budućih sustava

- Memorijski sustav
  - Aplikacije su sve više intenzivne s obzirom na podatke
  - Seljenje podataka limitira performanse i efikasnost
- Efikasnost (performanse i energija) → skalabilnost
  - Omogućava razvoj skalabilnijih sustava - nove aplikacije
  - Omogućava bolje korisničko iskustvo - novi usage modeli
- Predvidljivost i robusnost
  - Dijeljenje resursa i nepouzdanog hardvera = QoS problem
  - Predvidljivost performansi i QoS su primarna ograničenja

# Komercijalni primjeri asimetričnog dizajna

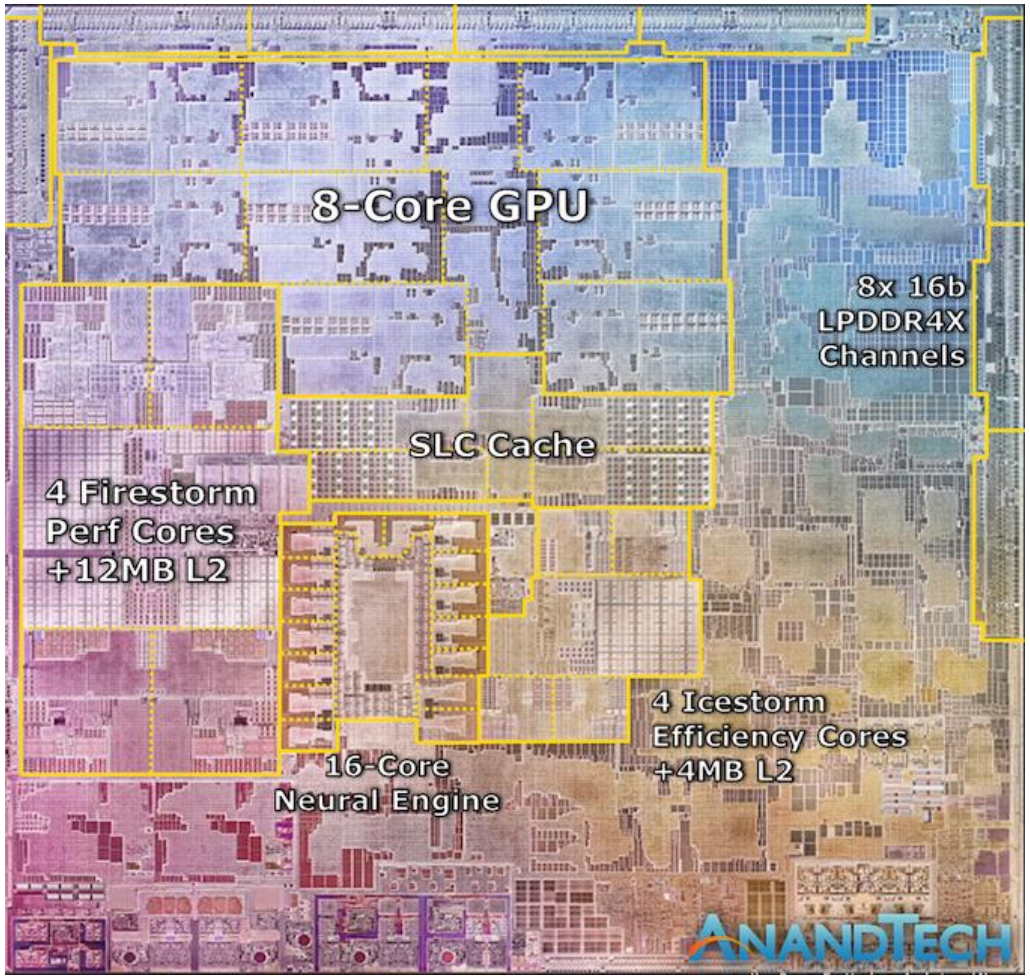
- Integrirani CPU+GPU procesori (npr. Intel procesori sa GPUom)
- CPU+bilo koji hardverski akcelerator (npr. mobilni telefon)
- Heterogeni multi-core sustavi
  - ARM big.LITTLE
  - IBM Cell
- CPU + FPGA sustavi (velika količina primjera)

# Povećanje heterogenosti u modernim sustavima




- Heterogeni agenti: Veliki i mali procesori, GPU, akceleratori
- Heterogene memorije: Brzi i spori DRAM, NVMe
- Heterogene sabirnice : podaci, kontrola, sinhronizacija

# Trenutne heterogene SoC arhitekture




Apple M1, 2021

# Apple M1


5 nanometer process
Machine learning accelerators
16-core
Neural Engine
11 trillion operations per second
Thunderbolt / USB 4 controller
Media encode and decode engines
M1
16 billion transistors
Up to
8-core GPU
8-core CPU
Advanced image signal processor
Secure Enclave
Unified memory architecture
Industry-leading performance per watt

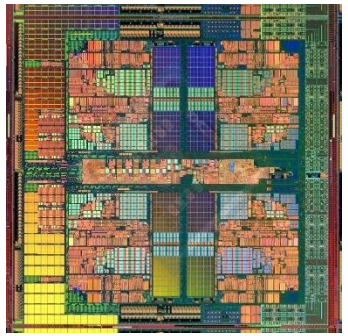
# Apple M2


Up to
24GB
LPDDR5 memory
High-performance media engine
40%
Faster Neural Engine
Up to
15.8 trillion
operations per second
16-core Neural Engine
Second-generation 5 nm technology
M2
ProRes
encode and decode
6K external display support
8-core CPU
Up to
10-core GPU
18%
Faster CPU
35%
Faster GPU
Over
20 billion
transistors
Industry-leading performance per watt
50%
More memory bandwidth
100GB/s
Memory bandwidth

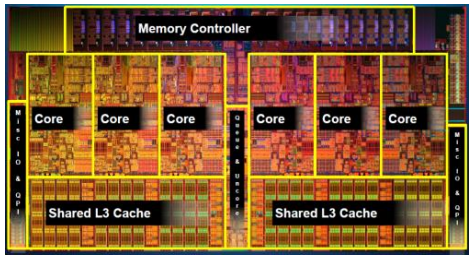
# Dizajn multi-core sustava: heterogeni princip

# "Puno" jezgri na čipu (manycore)

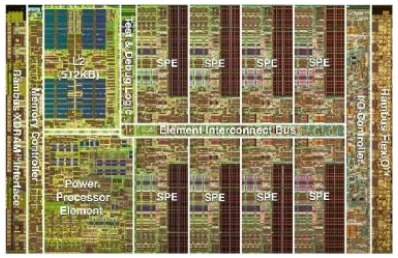
- Jednostavniji dizajn, troši manje struje nego jedna velika jezgra
- Široki paralelizam na čipu



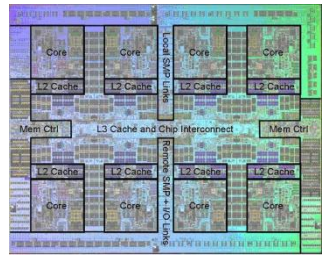
AMD Barcelona
4 cores



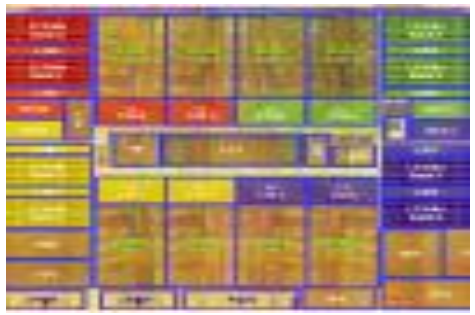

Intel Core i7
8 cores



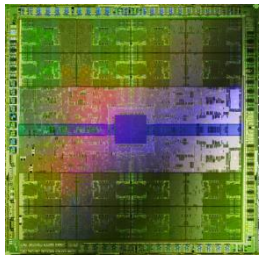

IBM Cell BE
8+1 cores



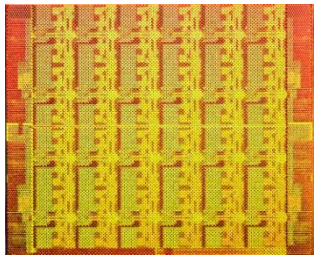

IBM POWER7
8 cores



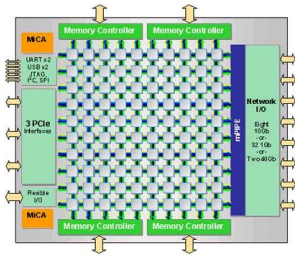
Sun Niagara II
8 cores



Nvidia Fermi
448 “cores”



Intel SCC
48 cores, networked




Tilera TILE Gx
100 cores, networked

# Sa puno jezgri na čipu

- Što želimo:
  - N puta više performansi ako koristimo N puta više jezgri kada paraleliziramo aplikaciju tako da radi na N jezgri
- Što dobivamo:
  - Amdahl-ov zakon (usko grlo u serijskom izvršavanju)
  - uska grla u paralelnom izvršavanju

# Problematika paralelizma

- Amdah-lov zakon
  - f: dio programa koji se može paralelizirati
  - N: broj procesora

$$\text{Overall Speedup} = \frac{\text{Old execution time}}{\text{New execution time}}$$

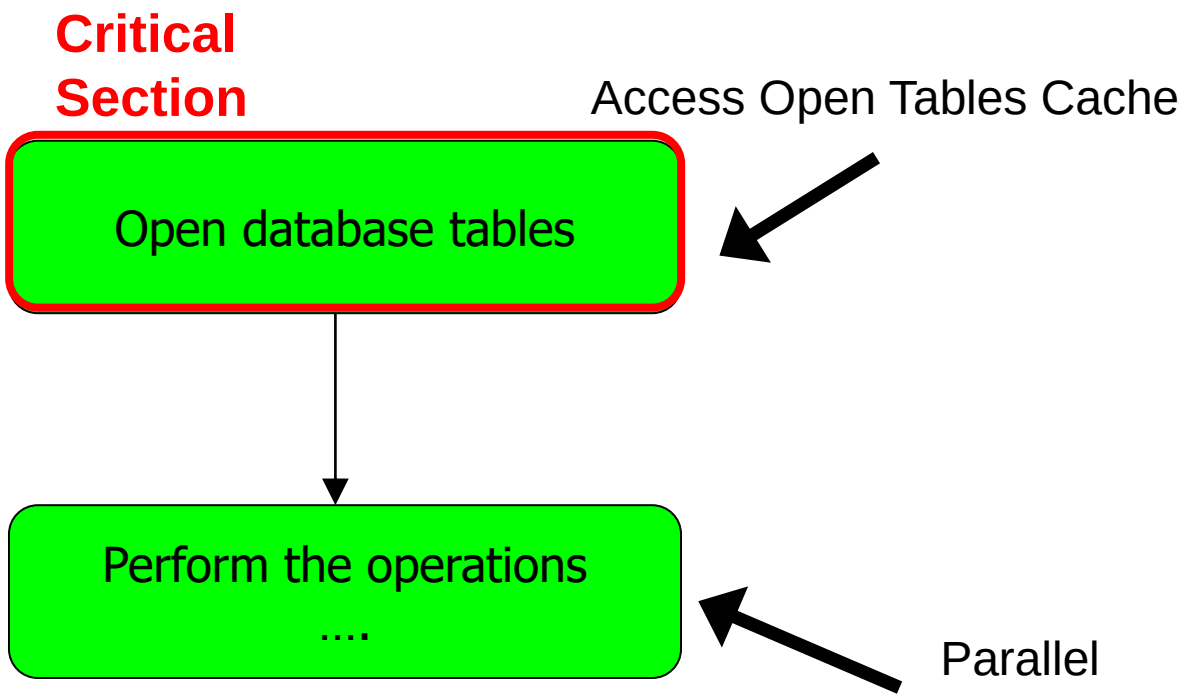
$$= \frac{1}{\left( 1 - \text{Fraction}_{\text{enhanced}} \right) + \left( \frac{\text{Fraction}_{\text{enhanced}}}{\text{Speedup}_{\text{enhanced}}} \right)}$$

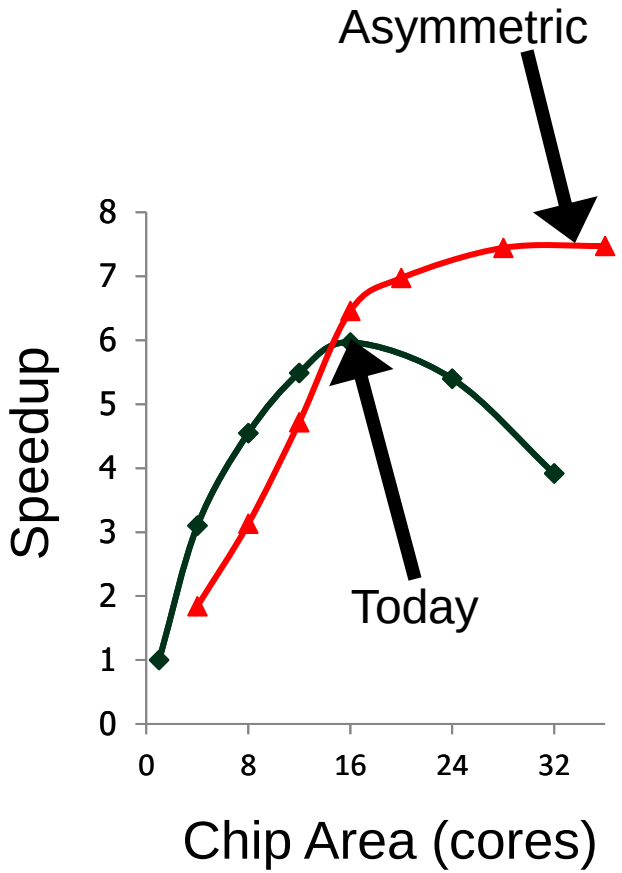
  - Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS 1967.
- Maksimalno ubrzanje limitirano je serijskim dijelom aplikacije - serijskim uskim grlom
- Paralelni dio aplikacije nikad nije savršeno paralelan
  - Overhead sinhronizacije (npr. mijenjanje djieljenih podataka)
  - Overhead neujednačenog opterećenja (nesavršenost u paralelizaciji)
  - Overhead dijeljenja resursa (natjecanje za resurse, za N procesora)

# Problem: Serijski dio koda

- Ne može se sve paralelizirati
- Razlozi:
  - sekvencijalni dijelovi koda (Amdah-lov serijski dio koda)
  - kritične sekcije
  - prepreke
  - limitirana mogućnost pipelininga
- Serijski dijelovi koda:
  - smanjuju performanse
  - smanjuju skalabilnost
  - troše energiju

# MySQL primjer

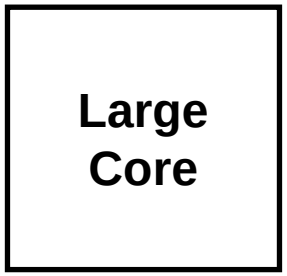


# Zahtjevi različitih dijelova koda

- Što želimo:
  - U serijskom dijelu koda - jednu veliku, brzu jezgru
  - U paralelnom dijelu koda - veću količinu malih jezgri
- Ove dvije želje su u direktnom konfliktu
  - ako imamo jednu veliku, brzu jezgru, ne možemo imati puno jezgri
  - mala jezgra je puno više energetski i prostorno efikasna od velike jezgre

# “Velike” vs. “Male” jezgre



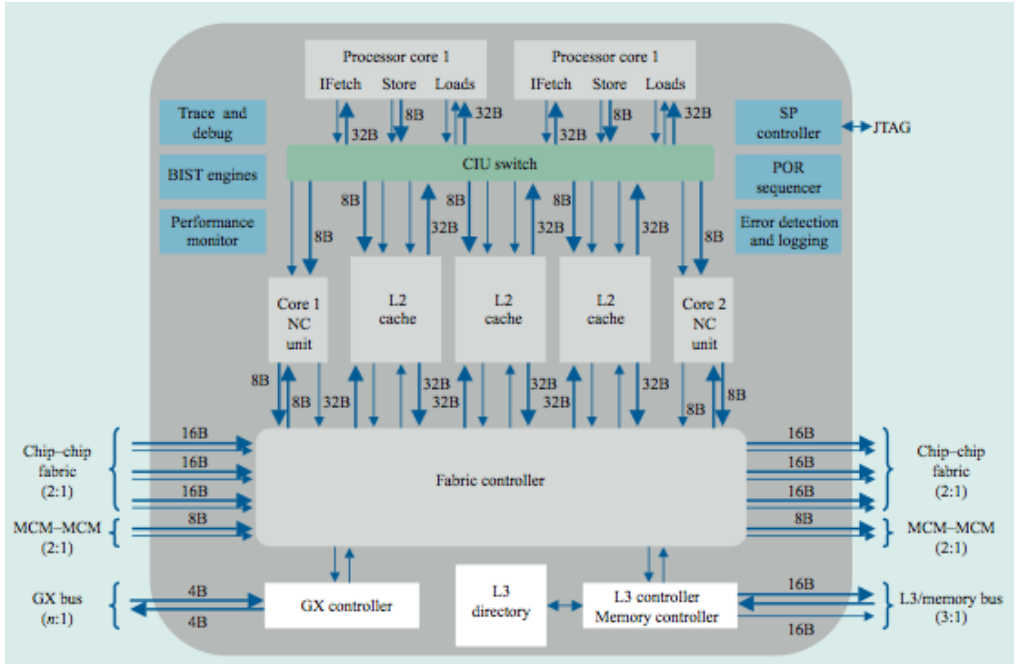

- *Out-of-order*
- *Wide fetch e.g. 4-wide*
- *Deeper pipeline*
- *Aggressive branch predictor (e.g. hybrid)*
- *Multiple functional units*
- *Trace cache*
- *Memory dependence speculation*

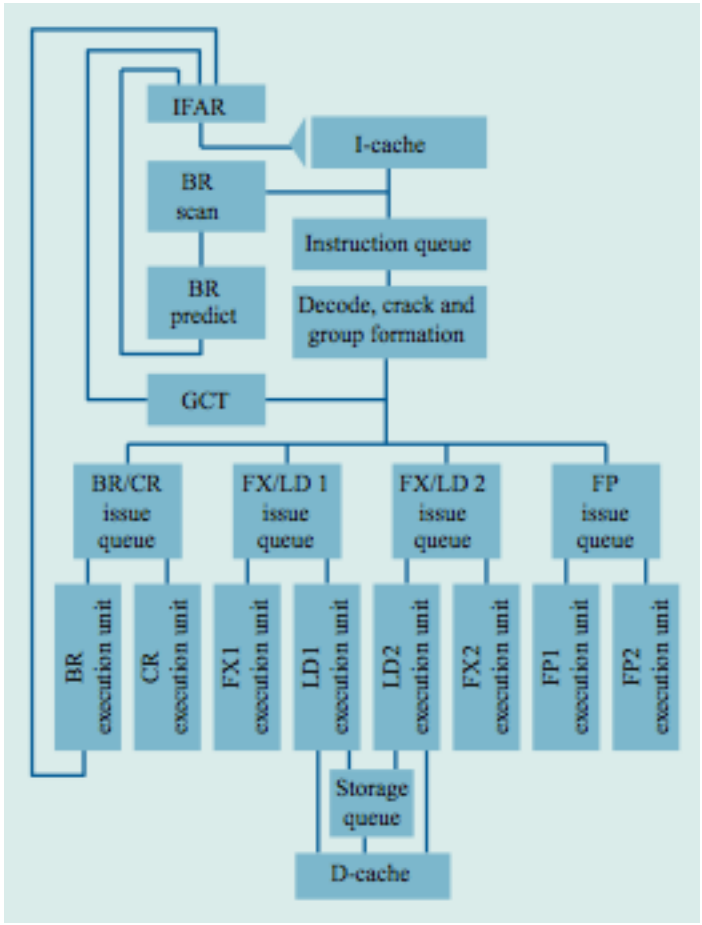

- *In-order*
- *Narrow Fetch e.g. 2-wide*
- *Shallow pipeline*
- *Simple branch predictor (e.g. Gshare)*
- *Few functional units*

Velike jezgre nisu efikasne:
npr. 2x više performansi za 4x veću površinu

# Velika jezgra: IBM POWER4

- Simetrični multi-core chip…
- Dvije velike jezgre



# IBM POWER4

- 2 jezgre, out-of-order execution
- 100-entry instruction window u svakoj jezgri
- 8-wide instruction fetch, issue, execute
- veliki, lokalni i globalni hybrid branch predictor
- 1.5MB, 8-way L2 cache
- Aggressive stream based prefetching

# IBM POWER5

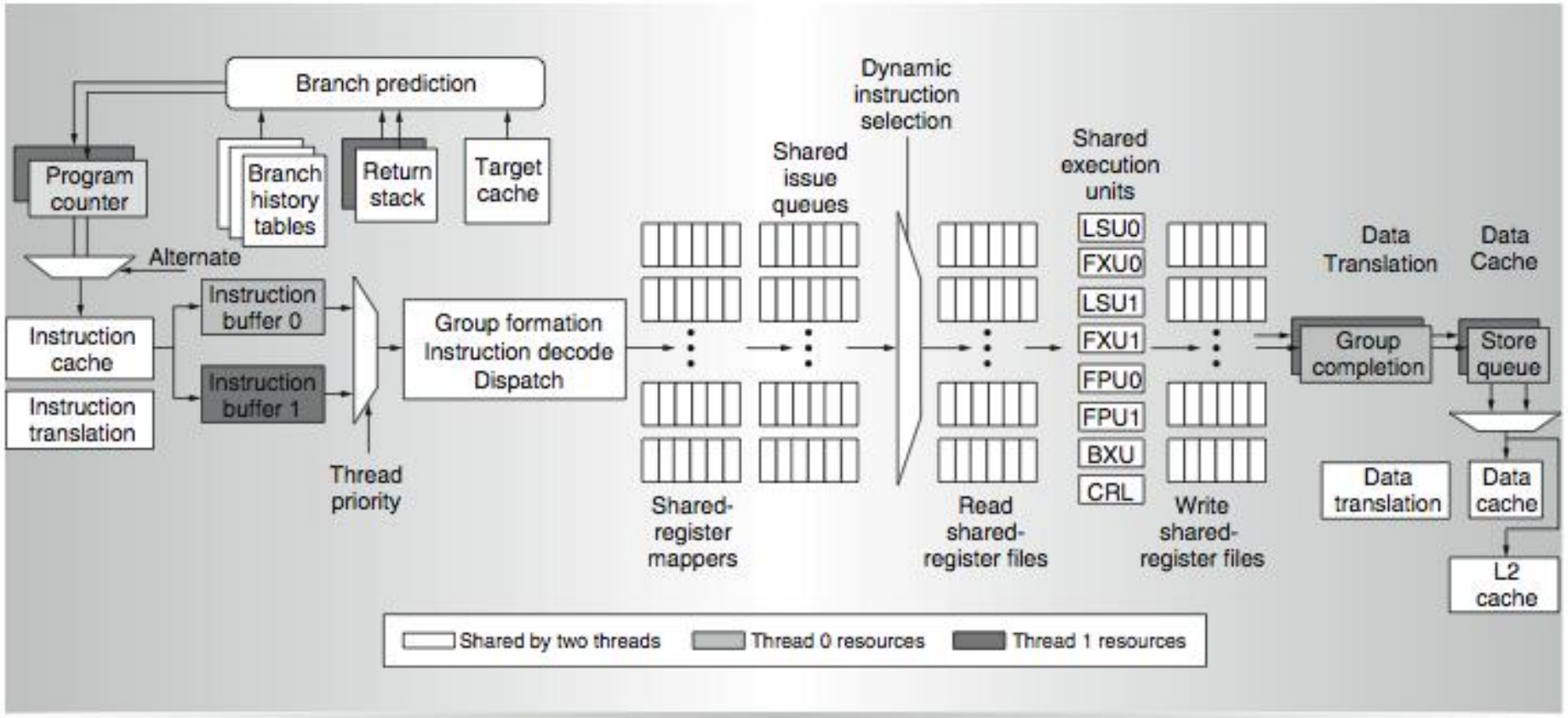
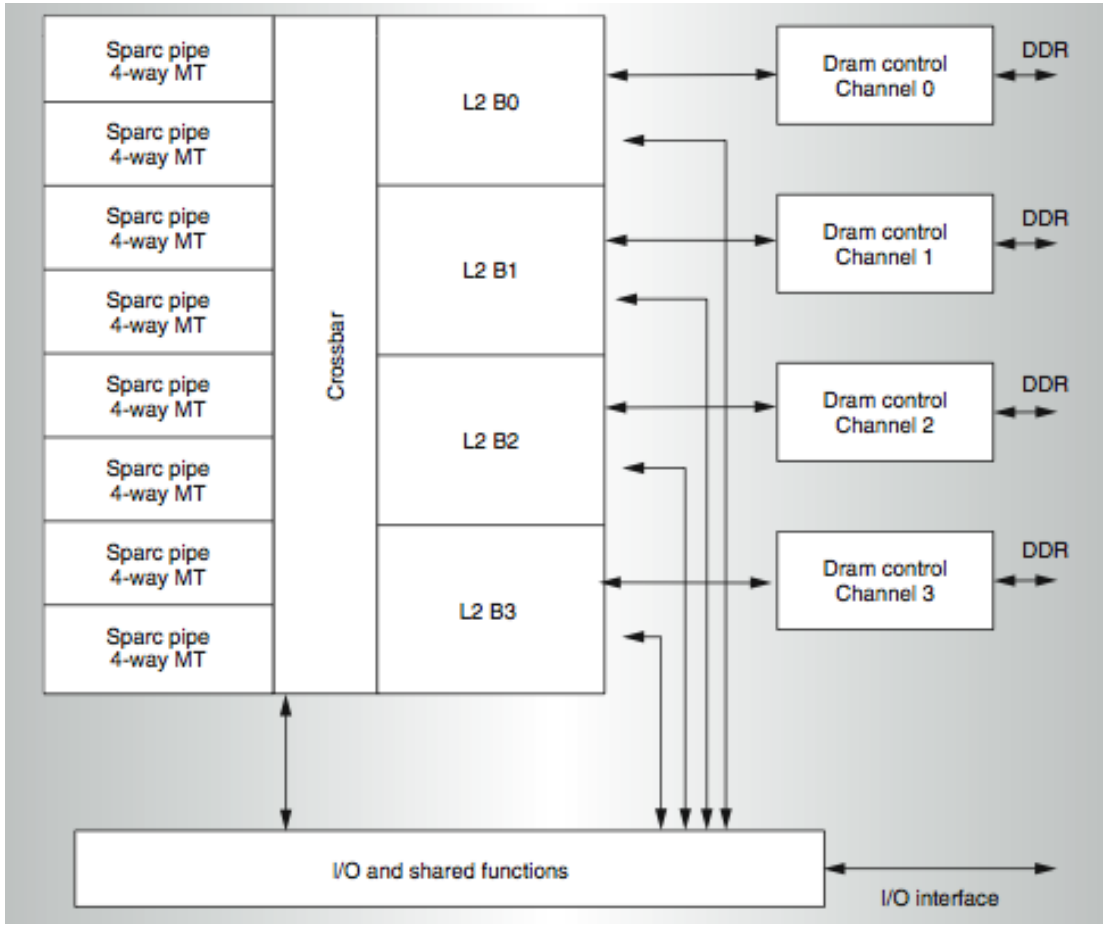


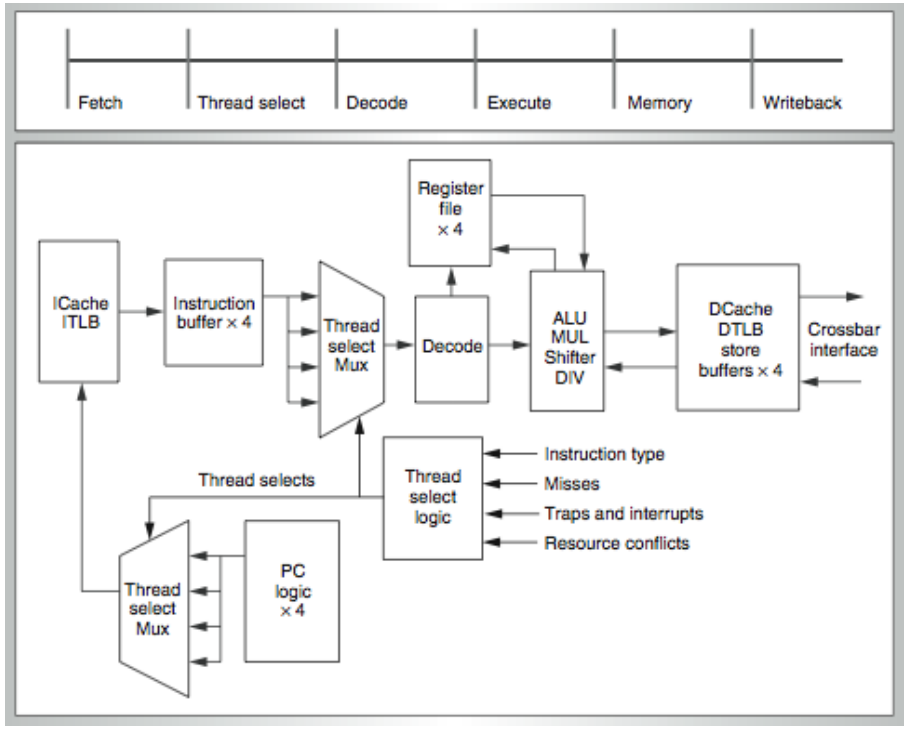

Figure 4. Power5 instruction data flow (BXU = branch execution unit and CRL = condition register logical execution unit).

# Mala jezgra: Sun Niagara (UltraSPARC T1)



# Niagara Core

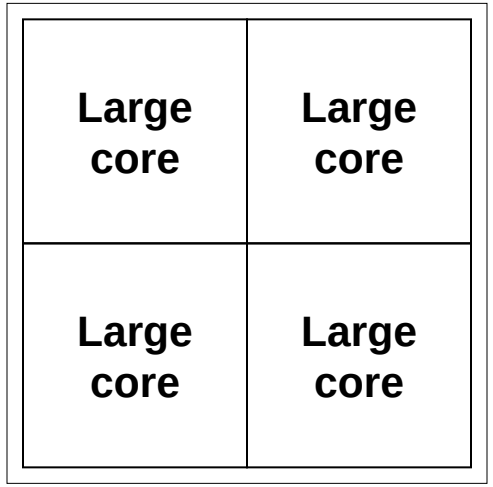
- četverojezgreni, 6-stage, in-order procesor
- round robin odabir threadova (osim kada se dogodi cache miss)
- dijeljeni FP unit



# Performance vs. Parallelism

- Pretpostavke:
    1. Manja jezgra ima budžet za površinu = 1 i performanse 1
    2. Veća jezgra ima budžet za površinu = 4 i performanse 2

# Pristup sa velikim jezgrama



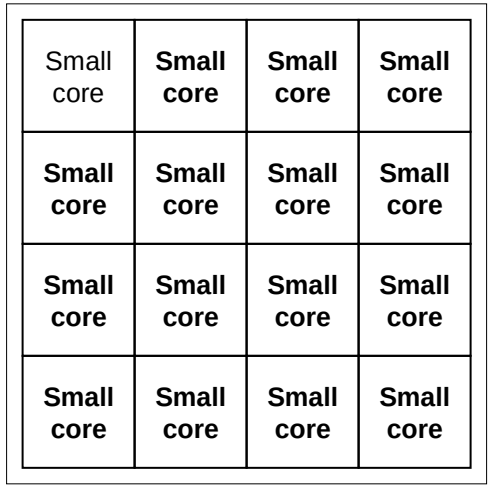

“Tile-Large”

- pločica sa više velikih jezgri
- IBM Power 5, AMD Barcelona, Intel Core2Quad, Intel Nehalem

+ visoke single-thread performanse, tj.serijske performanse

- loše performanse u paralelnom procesiranju

# Pristup sa malim jezgrama




“Tile-Small”

- na pločici je skupina malih jezgri
- Sun Niagara, Intel Larrabee, Tilera TILE (tile ultra-small)

\+ velika brzina izvršavanja paralelnog dijela koda

\- loše performanse serijskog dijela koda

# Da li možemo dobiti oboje?

- velike jezgre

  + visoke single-thread performanse, tj.serijske performanse

- loše performanse u paralelnom procesiranju

- male jezgre

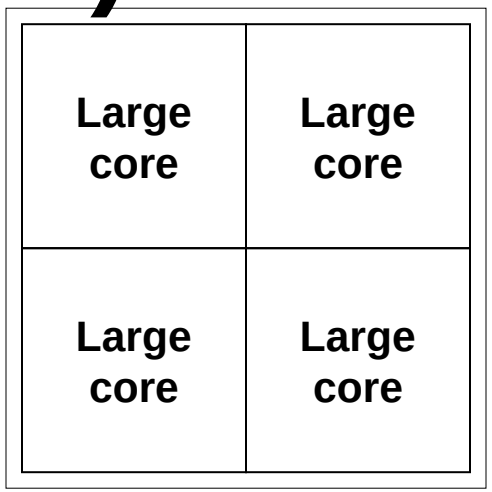
+ velika brzina izvršavanja paralelnog dijela koda

- loše performanse serijskog dijela koda

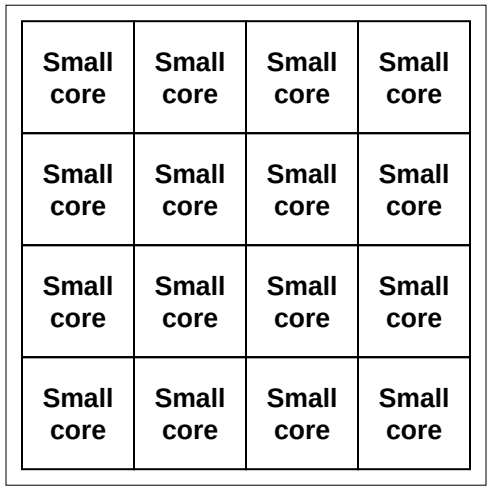
- Ideja: Obje vrste jezgri na jednom čipu - asimetrične performanse

# Asimetrični multi-core procesori

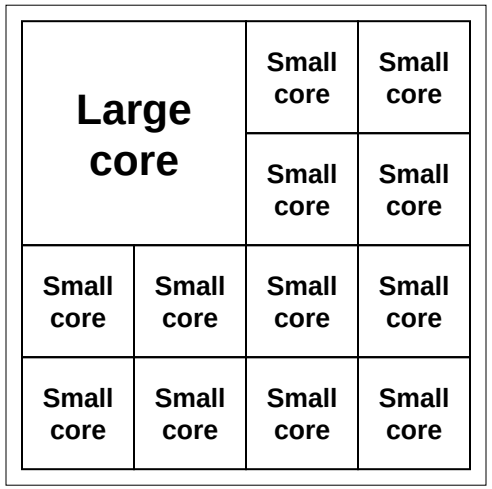
# Asymmetric Chip Multiprocessor (ACMP)




“Tile-Large”




“Tile-Small”



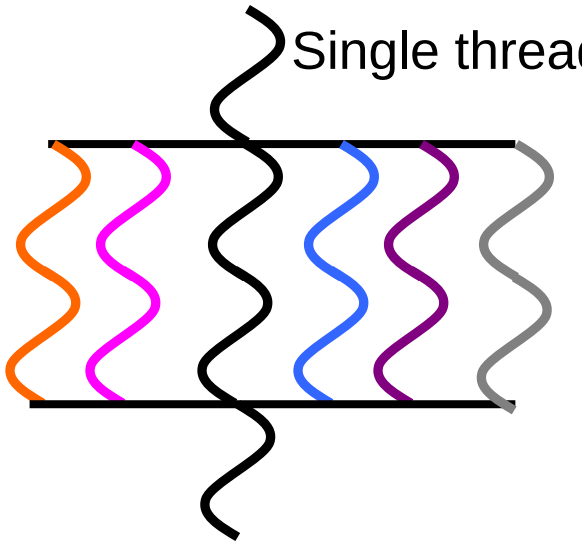

ACMP

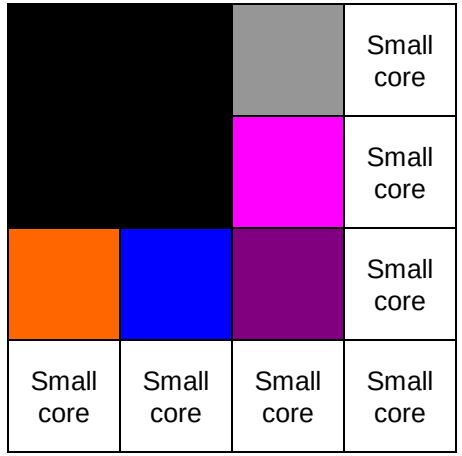
- Jedna velika i više malih jezgri

+ Ubrzavanje serijskog dijela kroz velike jezgre

+ Ubrzavanje paralelnog dijela kroz male jezgre i korištenje velike jezgre za veliku propusnost

# Accelerating Serial Bottlenecks

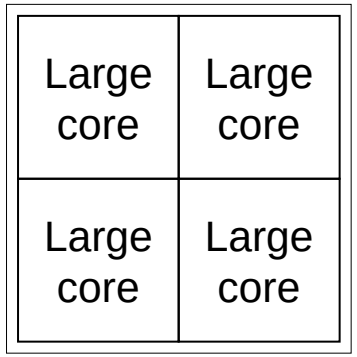


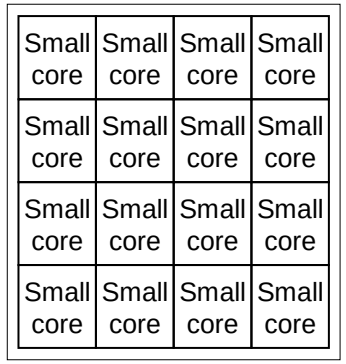

ACMP pristup

# ACMP performanse vs. paralelizam

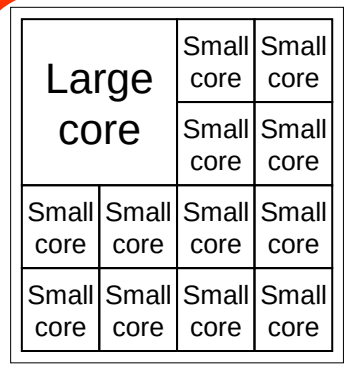
*Area-budget = 16 small cores*




“Tile-Large”




“Tile-Small”




ACMP

| | “Tile-Large” | “Tile-Small” | ACMP |
|---|---|---|---|
| Large Cores | 4 | 0 | 1 |
| Small Cores | 0 | 16 | 12 |
| Serial Performance | 2 | 1 | 2 |
| Parallel Throughput | 2 x 4 = 8 | 1 x 16 = 16 | 1x2 + 1x12 = 14 |

# Modifikacija Amdahl-ovog zakona

- Pojednostavljeni Amdahl-ov zakon za asimetrični multiprocessor
- Pretpostavke:
  - Serijski kod ide na veliku jezgru
  - Paralelni kod ide na male I velike jezgre
  - f: dio koda koji se da paralelizirati
  - L: broj velikih jezgri
  - S: broj malih jezgri
  - X: ubrzanje velike jezgre u odnosu na malu

$$\text{Ubrzanje} = \frac{1}{\frac{1 - f}{X} + \frac{f}{S + X*L}}$$

# Accelerating Parallel Bottlenecks

- Serializirano ili nebalansirano izvođenje u paralenom dijelu koda može imati benefita od velike jezgre
- Primjer:
  - Kritične sekcije u kojem postoji natjecanje za resurse
  - Paralelne sekcije kojima treba dulje da se završe
- Ideja: dinamički identificirati te dijelove koda I pokrenuti ih na velikoj jezgri

# Drugi primjer asimetričnosti

# Asimetričnost za energetsku efikasnost

- Kumar et al., “Single-ISA Heterogeneous Multi-Core Architectures: The Potential for Processor Power Reduction,” MICRO 2003.
- Ideja:
  - Implementirati više tipova jezgri na procesoru
  - Nadgledati karakteristike threada koji je pokrenut (npr. periodički uzorkovati potrošnju energije I performansi svake jezgre)
  - Dinamički odabrati jezgru koja daje najbolji odnos energija/performanse za određenu fazu
    - Najbolja jezgra – ovisi o metrici koju koristimo za optimizaciju

# Korištenje asimetrije za energetsku efikasnost

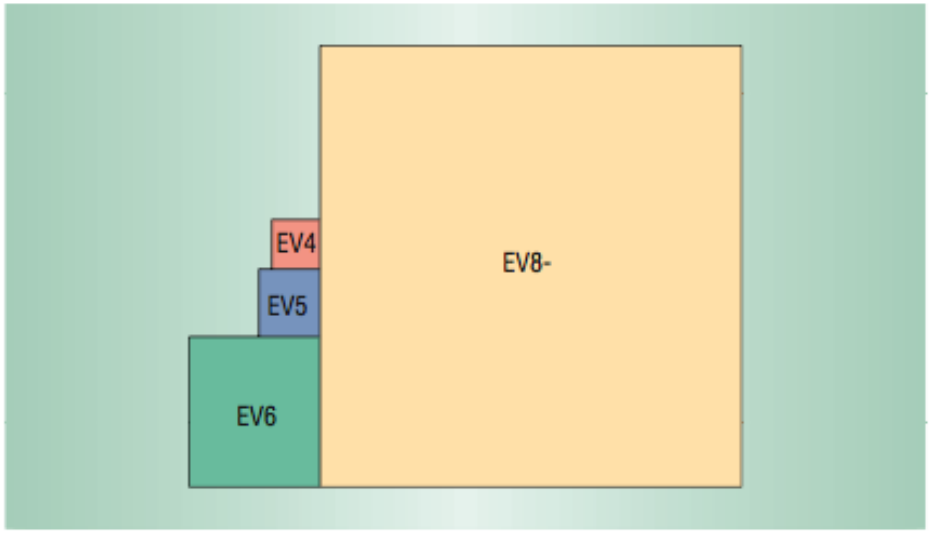
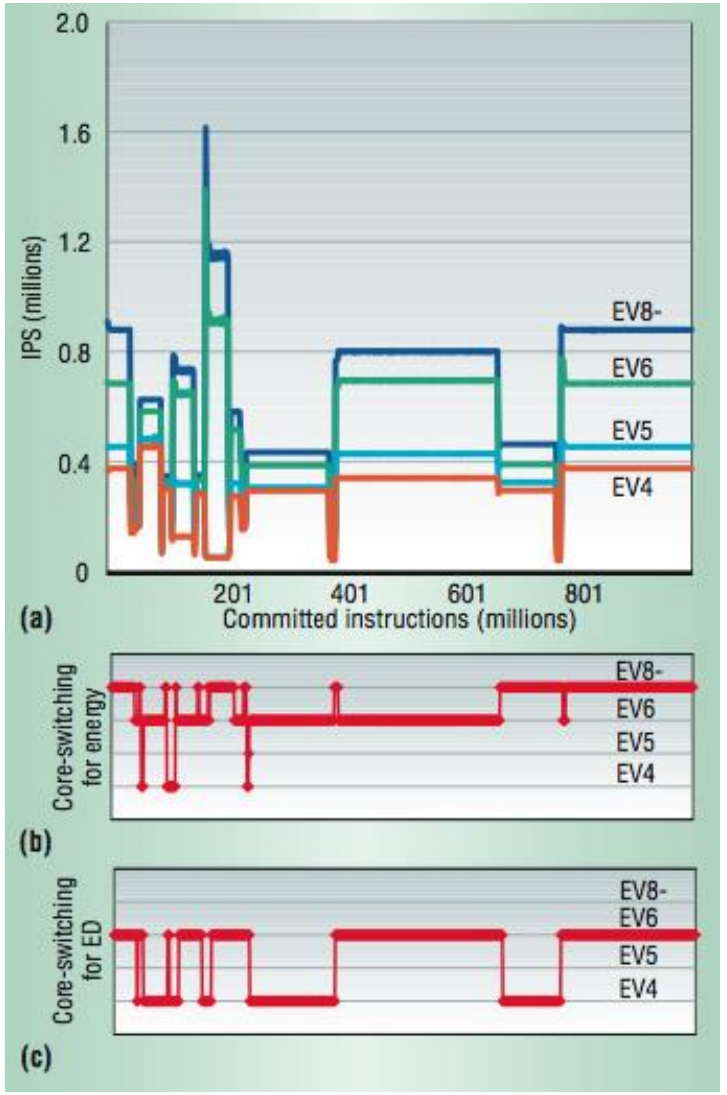



*Figure 1. Relative sizes of the Alpha cores scaled to 0.10 µm. EV8 is 80 times bigger but provides only two to three times more single-threaded performance.*

**Table 1. Power and relative performance of Alpha cores scaled to 0.10 µm. Performance is expressed normalized to EV4 performance.**

| Core | Peak power (Watts) | Average power (Watts) | Performance (norm. IPC) |
|---|---|---|---|
| EV4 | 4.97 | 3.73 | 1.00 |
| EV5 | 9.83 | 6.88 | 1.30 |
| EV6 | 17.8 | 10.68 | 1.87 |
| EV8 | 92.88 | 46.44 | 2.14 |



# Korištenje asimetrije za energetsku efikasnost

- Prednosti
  + Fleksibilnost
  + Možemo izvršavati operacije na jezgri koja je najbolja za te operacije (u smislu energetske potrošnje)

- Mane/problemi
  - Ako napravimo krivo predviđanje – izvršavanje na krivoj jezgri – lose performanse, pretjerana potrošnja energije ili oboje
  - Overhead zbog prebacivanja između jezgri
  - Trebamo napredno nadgledanje faze izvršavanja
    - Koje karakteristike nadgledati?
    - Kako odabrati jezgru?

# Kako postići asimetriju?

- Statički
  - Tip I snaga jezgri fiksirani u fazi dizajna
  - Dva pristupa za dizajn “bržih” jezgri:
    - Viša frekvencija
    - Kompleksnija, robustnija jezgra sa drugačijom mikroarhitekturom
  - Da li je statička asimetrija prirodna? (varijacije u fekvenciji čipa postoje ovako I onako)
- Dinamički
  - Tip I snaga jezgri se dinamički mijenja
  - Dva pristupa za dizajn “bržih” jezgri:
    - Dinamički podići frekvenciju (unutar zadane potrošnje snage)
    - Kombinirati manje jezgre kako bismo dobili kompleksniju, jaču jezgru
    - Možda postoje I drugi pristupi

# Asimetričnost kroz podizanje frekvencije

# Asimetričnost kroz podizanje frekvencije

- Statička
  - Varijacije u proizvodnom procesu znače da jezgre same po sebi mogu imati različitu frekvenciju
  - Namjerno dizajnirati jezgre tako da imaju različite frekvencije

- Dinamička
  - Annavaram et al., “Mitigating Amdahl’s Law Through EPI Throttling,” ISCA 2005.
  - Dinamičko skaliranje korištenjem napona I frekvencije

# EPI Throttling

- Cilj: Minimizacija vremena izvršavanja paralelnih programa kroz zadanu energetsku potrošnju
- Za najbolje performanse skalarne obrade I propusnosti, variramo količinu potrošene energije po instrukciji (EPI) bazirano na dostupnom paralelizmu
  - P = EPI •IPS
  - P = fixed power budget
  - EPI = energy per instruction
  - IPS = aggregate instructions retired per second
- Ideja: Za fiksirani budžet energije
  - Pokretati sekvencijalne zadatke na high-EPI procesoru
  - Pokretati paralelne zadatke na više low-EPI procesora

# EPI Throttling kroz dinamičko skaliranje frekvencije naponom

- DVFS: Dynamic voltage frequency scaling
- U fazama koje imaju nisku razinu paralelizma
  - Pokrenuti izvršavanje na malo jezgri sa visokom frekvencijom I potrošnjom
- U fazama koje imaju visoku razinu paralelizma
  - Pokrenuti izvršavanje na velikom broju jezgri sa niskom frekvencijom I potrošnjom

# Neke od mogućih EPI tehnika

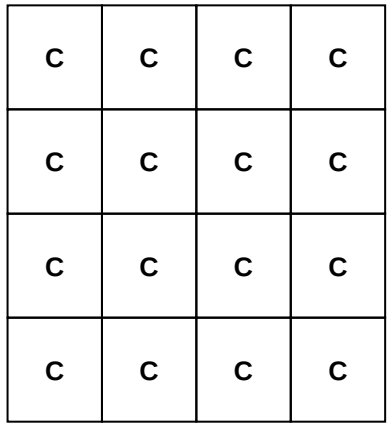
- Grochowski et al., “Best of both Latency and Throughput,” ICCD 2004.

| Method | EPI Range | Time to Alter EPI | Throttle Action |
|---|---|---|---|
| Voltage/frequency scaling | 1:2 to 1:4 | 100us (ramp Vcc) | Lower voltage and frequency |
| Asymmetric cores | 1:4 to 1:6 | 10us (migrate 256KB L2 cache) | Migrate threads from large cores to small cores |
| Variable-size core | 1:1 to 1:2 | 1us (fill 32KB L1 cache) | Reduce capacity of processor resources |
| Speculation control | 1:1 to 1:1.4 | 10ns (pipeline latency) | Reduce amount of speculation |

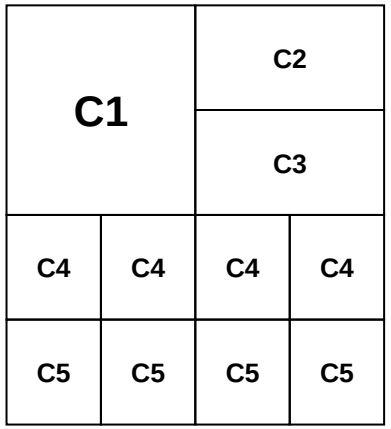
# Podizanje frekvencije male vs velike jezgre

- Podizanje frekvencije imaju svi Intelovi procesori od Nehalem jezgre na dalje, I npr. IBM POWER7
- Prednosti
  + jednostavna implementacija, ne trebamo redizajnirati ništa
  + paralelna propusnost ne pada kada je razina thread-level paralelizma visoka
  + čuva lokalnost threada koji radi na jezgri podignute frekvencije
- Mane
  - Ako je thread ograničen memorijom, nema ubrzanja
  - Ne smanjuje količinu ciklusa po instrukciji (CPI)
  - Mijenjanje frekvencije/napona može trajati dulje nego prebacivanje na veću jezgru

# Asimetričnost omogućava prilagodbu




Symmetric




Asymmetric

- Simetrično: One size fits all
  - Potrošnja energije i performanse nisu optimalni za različite aplikacije
- Asimetrično: Prilagodba
  - Potreba za procesiranjem varira s obzirom na aplikaciju
  - Pokretanje koda na resursu koji je najbolji fit

# Puno pitanja za buduća istraživanja

- Kako dizajnirati asimetrične komponente?
  - Fiksno, raspodijeljeno, sa mogućnošću rekonfiguraciju?
  - Kakvi tipovi asimetričnosti? Koje tehnologije?
- Kakav nadzor provoditi u hardveru I softveru?
  - Za otkrivanje faze I potreba trenutnog izvršavanja
- Kako dizajnirati feedback/kontrolnu petlju između komponenti I sistemskog softvera?
- Kako dizajnirati softver da automatski upravlja resursima?
  - Praćenje ponašanja threadova, odabir best-fit komponenti za workload

# Specifični use-case: HPC, I

- "Ever since the invention of supercomputers in the 1960s, science world has been one of the key customers on that market. As technology, programming and computational models improved, another type of supercomputer was introduced, and its subsequent application was called HPC (High Performance Computing). High Performance Computing is the application of supercomputers to computational problems that are either too large for standard computers or would take too long."

Dakic et al., “Optimizing Kubernetes Performance, Efficiency and Energy Footprint in Heterogenous HPC Environments,” DAAAM 2021. (remaining slides)

# Specifični use-case: HPC, II

- "By definition, heterogenous computing means mixing different types of cores within the limits of a single node. We can use this approach in a variety of different ways, like:
  - by using CPU and GPU in the same node (for example, x86 CPU and NVIDIA Tesla GPU).
  - by using CPU and ASIC in the same node (for example, x86 CPU coupled with Intel eASIC or Xilinx UltraScale).
  - by using different CPU architectures in the same node (for example, AMD SkyBridge)".

# Heterogeni pristupi podržani hardverom

- "There are multiple different technologies that are prominent - on desktop, server and HPC markets. Some common examples include:
  - GPGPU (General-Purpose Graphics Processing Unit) technologies like CUDA (Compute Unified Device Architecture) and OpenCL (Open Computing Language) on various hardware platforms like NVIDIA/AMD GPUs paired with x86 processors.
  - Intel Xeon Phi and similar accelerators, that can take advantage of concepts like OpenCL, OpenMP, C/C++, Fortran etc.
  - Xilinx and similar FPGA/ASICs, that can also take advantage of various concepts like OpenCL, HLS etc.”
- “All these solutions can be mixed and matched and used side-by-side with other solutions, like Open MPI (Open Message Passing Interface), Intel MPI, MPICH (Message-Passing Interface Chameleon) for distributed parallel computing, which fits well with the concept of HPC. Idea behind these technologies is quite straightforward – to run code on multiple processors and use one of these solutions via library calls."

# Zašto je ovo zanimljivo?

- “x86-based ecosystem has been well-developed over the past 40+ years, which means that a lot of workloads can be handled by it.”
- “ARM-based ecosystem has been focused more on non HPC markets in the past However, the newly developed 64-bit architecture along with their respective SoCs will have a large impact on the CPU performance and memory bandwidth of these ARM processors in HPC domain.”
- “RISC-V based ecosystem has been focused on open-sourcing CPU technology via open instruction set and could be the “Linux of CPU design” if everything goes well.”
- “GPUs are very fast for specific types of workloads.”
- “ASICs are even faster for specific types of workloads.”

# Zašto, prvi dio?

| Hardware | Kubernetes node specifications | | | |
|---|---|---|---|---|
| | *Node 1* | *Node 2* | *Node 3* | *Node 4* |
| CPU type | 2xXeon E5-2630L | 2xXeon E5-2450L | 2xXeon E5-2680 | 1xCortex A72 (ARM v8) |
| Core number /SMT core number | 12/24 | 16/32 | 16/32 | 4/4 |
| Base frequency | 2 GHz | 1.8 GHz | 2.7 GHz | 1.8 GHz |
| L1 cache (per core) | 64 KB | 64 KB | 64 KB | 80 KB |
| L2 cache | 256 KB per core | 256 KB per core | 256 KB per core | 1 MB shared (with GPU) |
| L3 cache (shared) | 15 MB | 20 MB | 20 MB | - |
| Memory | 96 GB | 96 GB | 96 GB | 4 GB |
| Memory frequency | 1333 MHz | 1333 MHz | 1333 MHz | 3200 MHz |
| Average power consumption | 220 W | 230 W | 250 W | 13W |

Table 1. Nodes used for preliminary testing

# Zašto, drugi dio?

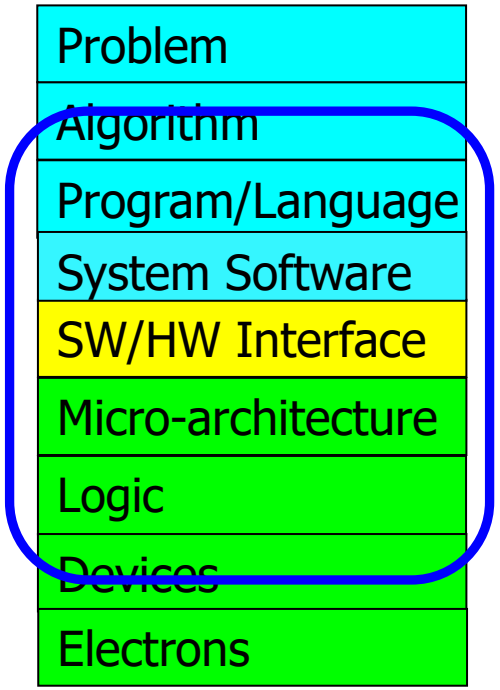
| Hardware | Kubernetes node performance | | | |
|---|---|---|---|---|
| | *Node 1* | *Node 2* | *Node 3* | *Node 4* |
| Sysbench single core (events per second) | 643.89 | 593.16 | 815.52 | 1787.74 |
| Sysbench multicore (events per second) | 1299.85 | 1196.87 | 1630.47 | 3574.08 |
| Sysbench memory (total time) | 9.55366 | 10.0014 | 7.57572 | 10.0002 |
| Stress-ng single core (bogo ops) | 12258.4 | 11274.4 | 14836.4 | 1253 |
| Stress-ng multi core (bogo ops) | 24286.8 | 22398.0 | 29626.8 | 2134 |
| HPL (Gflops) | 2.01437 | 1.76892 | 2.47621 | 10.9701 |
| RandomAccess (GUP/s) | 0.0622563 | 0.0471568 | 0.0416779 | 0.0075283 |
| FFT (Gflops) | 1.99787 | 1.74458 | 2.06731 | 0.186364 |
| STREAM (GB/s) | 6.19918 | 5.35783 | 6.1504 | 0.537999 |
| PTRANS (GB/s) | 1.59295 | 1.35349 | 1.70072 | 0.130684 |
| DGEMM (Gflops) | 1.52678 | 1.32662 | 1.58446 | 4.19122 |
| HPL time (s) | 331.03 | 376.96 | 269.29 | 60.7851 |
| HPCG (Gflops) | 1.48279 | 1.37273 | 1.85292 | 0.100408 |

Table 2. Node performance

# Zaključak

- Budućnost je heterogena
- Budućnost je hibridna
- Budućnost je (vrlo vjerojatno) manycore
- Budućnost je u “vrlo bliskoj integraciji”
- Budućnost je cross-design.

To achieve the highest energy efficiency and performance:
**we must take the expanded view**
of computer architecture




**Co-design across the hierarchy: Algorithms to devices**

**Specialize as much as possible within the design goals**

# Hvala na pažnji!