MODALITĂŢI DE EVALUARE A PERFORMANTEI SISTEMELOR DE CALCUL

TOPICI de urmărit în cadrul cursului

METRICI DE EVALUARE A SISTEMELOR DE CALCUL
ALEGEREA METRICILOR.
LEGEA LUI AMDAHL[1]
SIMULATOARE SOFTWARE DEDICATE MICROARHITECTURILOR DE CALCUL.
METODOLOGII DE SIMULARE.
TOOLS-URI SI BENCHMARK-URI STANDARDIZATE.
ECUATIA DE STABILIRE A PERFORMANŢEI PROCESORULUI (CPU)

METRICI DE EVALUARE A SISTEMELOR DE CALCUL: *PERFORMANŢA DE PROCESARE*

Ca şi architect de microprocesoare scopul principal îl constituie realizarea unor compromisuri între următoarele METRICI: Performanţă de procesare, Putere disipată, Arie de integrare, Complexitate, Cost, Suportul oferit aplicaţiei, Funcţionalitate, Compatibilitate între modele aparţinând aceleaşi clase, Fiabilitate

Odată cu stabilirea unui set de cerinţe funcţionale ale sistemului, arhitectul de microprocesoare trebuie să optimizeze procesul de proiectare. Optimalitatea în proiectare depinde de metrica aleasă. Modificările realizate în ultima decadă în spaţiul aplicaţiilor cu sisteme de calcul au influenţat în mod dramatic rolul metricilor.

- **Sistemele desktop** sunt analizate (măsurate) prin prisma utilizatorilor individuali urmărindu-se **optimizarea raportului cost-performanţă**. Procesoarele sistemelor desktop sunt complexe, cu execuţie *out-of-order* şi *speculativă* a instrucţiunilor. Extrag paralelismul atât la nivelul instrucţiunilor (ILP) cât şi la nivelul firelor de execuţie (TLP).
- **Serverele** sunt optimizate din punct de vedere al **fiabilităţii, scalabilităţii** şi al **raportului cost - performanţă globală măsurată în tranzacţii** (operaţii) **per unitate de timp** (secundă, oră)
- **Sistemele dedicate** sunt optimizate din punct de vedere al **costului**, **ariei de integrare** şi al **puterii consumate**. Procesoarele ***embedded*** sunt mai simple, cu execuţie *in-order* a instrucţiunilor, dedicate în general aplicaţiilor de timp real. Nu conţin structuri de predicţie[2] a ramificaţiilor de program[3] => nu există execuţie speculativă, nu (prea) conţin *cache*-uri, trebuie evitat *Worst Case Execution Time* (sau să fie egal cu cazul general), conţin anumite instrucţiuni *sqrt*, dedicate multimedia, etc. În acest sens se va urmări ulterior în cadrul acestui curs diferenţele specifice dintre benchmark-urile de uz general (suita SPEC) şi benchmark-urile embedded (suita EEMBC).

Un sistem desktop (X) este mai performant decât altul (Y) dacă execută acelaşi program într-un timp (T_X) mai mic decât cel de-al doilea (T_Y).

---

[1] Legea scalabilităţii totale sau întâlnită şi sub denumirea legea randamentelor în scădere (*diminishing returns*)
[2] Predictoarele hardware sunt "*Power hungry and too large!*"
[3] Există variante cu predicţie statică de tip BTFNT

- se urmăreşte **reducerea timpului de execuţie** (sau **de răspuns –** timpul scurs între începutul şi sfârşitul unui eveniment).
- Timpul de execuţie (***latenţa***) reprezintă **inversul** (**reciproca**) performanţei de procesare
- Dacă (X) este de *"**n**"* ori mai performant decât (Y) atunci:

$$n = \frac{\text{Execution time}_Y}{\text{Execution time}_X} = \frac{\frac{1}{\text{Performance}_Y}}{\frac{1}{\text{Performance}_X}} = \frac{\text{Performance}_X}{\text{Performance}_Y}$$

Un server (SX) este mai performant decât altul (SY) dacă execută mai multe tranzacţii (operaţii, programe – *tasks, jobs*) în acelaşi interval de timp. Cantitatea de tranzacţii (operaţii, programe – *tasks, jobs, work*) efectuată într-un interval de timp bine stabilit se defineşte ***throughput*** sau **performanţa sistemelor server.**

- În cazul serverelor nu se urmăreşte reducerea timpului de execuţie ci **creşterea performanţei** (execuţia unui număr mai mare de operaţii în acelaşi interval de timp).
- **Throughput (taskuri/secundă)** vs. **Latenţa (secunde/pentru un task)**

Cea mai directă definiţie a **timpului de execuţie** o reprezintă "*latenţa de terminare a unui task, incluzând accesele la disc, la memorie, activităţi cu dispozitive periferice, alte interacţiuni cu sistemul de operare*". În cadrul sistemelor server, caracterizate de multiprogramare, procesorul (CPU) poate executa un alt program în timp ce aşteaptă finalizarea operaţiilor de I/O (citire / scriere din fişier, etc) nefiind necesară în acest caz minimizarea timpului de execuţie al task-ului principal.

- **CPU_time** – timp folosit exclusiv de procesor pentru calcul intern dintr-un singur task, neincluzând timpul de aşteptare pentru finalizarea operaţiilor I/O sau pentru execuţia altor task-uri.
- **CPU_time** – nu este sesizat de utilizatorul sistemului. Acesta vede întreg *timpul de execuţie*.
- **CPU_time** – poate fi descompus în:
  - **User_CPU_time** = timp petrecut cu instrucţiunile programului. Focalizarea se face pe această metrică în evaluarea şi optimizarea microarhitecturilor de calcul de pe parcursul acestui curs.
  - **System_CPU_time** = timp petrecut în cadrul sistemului de operare pentru execuţia diverselor rutine solicitate prin program (accese la periferice, memoria video, întreruperi software, etc). Depinde de fiecare SO pe care se execută task-ul respectiv. Poate reprezenta până la 35% din *CPU_time***.**

METRICI DE EVALUARE A SISTEMELOR DE CALCUL: DISPONIBILITATEA, FIABILITATEA, DEPENDABILITATEA

Pentru o mare perioadă de timp, majoritatea arhitecţilor de sisteme de calcul au urmărit **un singur scop principal**: **performanţa**. **Creşterea numărului de tranzistori care comutau tot mai rapid**, predicţionată de Gordon Moore prin legea care îi poartă numele, a fost transformată de proiectanţii de arhitecturi de calcul în creşterea remarcabilă a performanţei.

Reversul medaliei: Legea lui Moore a condus la apariţia conceptului de calculatoare omniprezente, răspândirea de dispozitive tot mai mici, dar care sunt caracterizate de o mai scăzută **dependabilitate** datorată de prezenţa defectelor fizice.

Scopul unui sistem de calcul tolerant la defecte este de a **funcționa în siguranță / funcționa corect** (**„*safety* and *liveness*”**) **în ciuda existenței / apariției unor erori / defecte**.
A *safe* computer never produces an incorrect user-visible result.

- ⇨ **If a fault occurs, the computer hides its effects from the user.**
- ⇨ **Safety alone is not sufficient, however, because it does not guarantee that the computer does anything useful.** A computer that is disconnected from its power source is safe – it cannot produce an incorrect user-visible result – yet it serves no purpose.

A *live* computer continues to make forward progress, even in the presence of faults. *Ideally, architects design computers that are both safe and live, even in the presence of faults*. However, even if a computer cannot provide liveness in all fault scenarios, maintaining safety in those situations is still extremely valuable.

It is preferable for a computer to stop doing anything rather than to produce incorrect results. An often used example of the benefits of safety, even if liveness cannot be ensured, is an Automatic Teller Machine (ATM). **In the case of a fault, the bank would rather the ATM shut itself down instead of dispensing incorrect amounts of cash.**

- **DISPONIBILITATEA** – fracțiunea din timpul total în care sistemul este disponibil pentru execuția oricărei aplicații:

$$Availability = \text{Mean Time To Fail} / (\text{Mean Time To Fail} + \text{Mean Time to Repair})$$

Disponibilitatea unui system la momentul *t* este egală cu probabilitatea ca sistemul să funcţioneze corect la momentul *t*.

- ***FIABILITATEA* (*reliability / dependability*) – unui sistem la momentul *t* este egală cu probabilitatea ca sistemul să funcţioneze corect de la momentul zero până la momentul *t*.**
  - Fiabilitatea este probabil cea mai cunoscută metrică și un termen frecvent folosit, dar este rareori o metrică potrivită pentru arhitecții de sisteme de calcul.
  - Cu excepţia cazurilor în care o defecțiune a sistemului este catastrofică (ex. *aviaţie*, *automotive*, etc.), **fiabilitatea este o metrică mai puțin utilă decât disponibilitatea**.
  - Pentru servere este **la fel de importantă disponibilitatea ca şi performanța**. Pierderea financiară a unei companii cotate la bursă în urma indisponibilității serverelor proprii este de peste ***$6,450,000 / oră***.
  - **Mean Time Between Failures** (MTBF)
    - măsoară timpul mediu între două căderi succesive ale sistemului (cât timp serverul este utilizabil)
    - Pentru harddiscul unui sistem desktop, uzual MTBF=500.000 de ore (~57 ani).
  - **Mean Time to Repair** (MTTR)
    - măsoară timpul mediu necesar revenirii dintr-o cădere a sistemului

- Pentru un server cu sarcini multe MTTR 0 ore (datorită redundanței oferite prin dublarea numărului de hardiscuri şi copierea informației în două locuri diferite)

Soluții pentru creşterea fiabilității sistemelor

- Redundanța
  - informațională – date de verificare suplimentară: *check bits* (paritate, coduri corectoare de erori)
  - hardware
- Mecanisme de check-pointing (salvarea stării sistemului la un moment dat şi reluarea execuției din acest punct)
- Programe antivirus

DEPENDABILITATEA

- În aplicațiile inginerești dependabilitatea reprezintă o metrică a disponibilității, fiabilității şi suportului de mentenanță a sistemelor. De asemenea, aceasta poate cuprinde şi mecanisme menite să mențină şi să crească fiabilitatea sistemelor.
- Prin cuvinte simple, sistemele dependabile (*dependable systems*) sunt sisteme de **securitate critice**. Se numesc astfel întrucât viața noastră poate depinde de acestea. În cazul sistemelor de securitate critice **timpul maxim de execuție al unei operații** (WCET – *worst case execution time*) trebuie să fie limitat şi cunoscut, astfel încât timpul de reacție poate fi asigurat atunci când sunt necesare acțiuni critice.
- **Dependabilitatea** poate fi analizată cunoscând trei categorii de elemente:
  - **Atribute** – Modalități de evaluare a dependabilității unui sistem
    - **Disponibilitatea** – efectuării corecte a unui serviciu
    - **Fiabilitatea** – continuitatea unui serviciu care funcționează corect
    - **Siguranță** – absența (evitarea) unor consecințe catastrofale asupra utilizatorului (utilizatorilor) şi a mediului
    - **Integritate** – absența (evitarea) unei modificări necorespunzătoare / alterări a sistemului
    - **Mentenabilitate**[4] – proprietate a unui produs de a fi întreținut şi reparat cu uşurință

[4] **Maintenance time** is defined as **the time between the start of the incident and the moment the system is returned to production** (i.e. how long the equipment is out of production). This includes notification time, diagnostic time, fix time, wait time (cool down), reassembly, alignment, calibration, test time, back to production, etc.

- **<u>Amenințări (pericole)</u>** – sunt tipuri de evenimente / situații care pot afecta funcționalitatea corectă a unui sistem cauzând o scădere a gradului de dependabilitate. Sunt în general trei astfel de categorii:
  - **Defecte**: A fault – "**o lipsă / greșeală**" (which is usually referred to as a *bug* for historic reasons) is a defect in a system. The presence of a fault in a system **may or may not lead to a failure (cădere a sistemului).** For instance, although a system may contain a fault, its input and state conditions may never cause this fault to be executed so that an error occurs; and thus that particular fault never exhibits as a failure.
  - **Erori**: An error is **a discrepancy between the intended behaviour of a system and its actual behaviour inside the system boundary**. Errors occur at runtime when some part of the system enters an unexpected state due to the activation of a fault. Since errors are generated from invalid states they are hard to observe without special mechanisms, such as debuggers or debug output to logs.
  - **Avarie (pană, rateu, insucces)**: A failure is an instance in time when a system displays behaviour that is contrary to its specification. An error may not necessarily cause a failure, for instance an exception may be thrown by a system but this may be caught and handled using fault tolerance techniques so the overall operation of the system will conform to the specification.
- **<u>Mijloace de îmbunătățire a dependabilității sistemelor</u>**
  - **Preîntâmpinare**
  - **Eliminare (îndepărtare)**
  - **Predicție (Anticipare)**
  - **Toleranță la defectări**

METRICI DE EVALUARE A SISTEMELOR DE CALCUL: <u>PUTEREA DISIPATĂ</u>

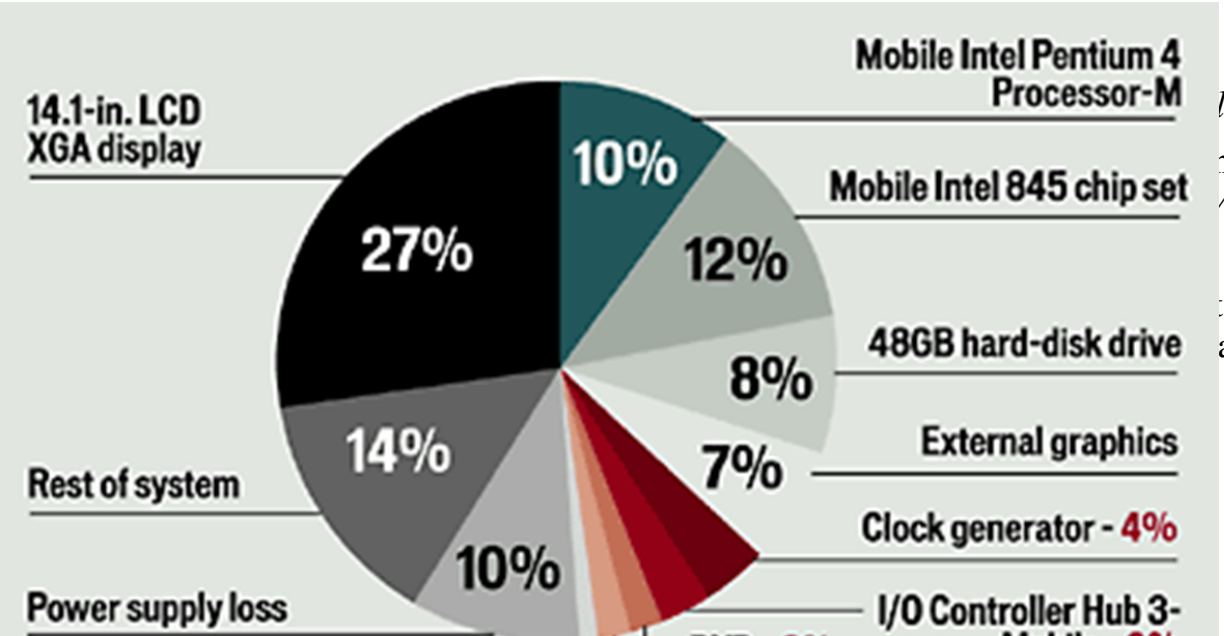
- Consumul de putere şi disiparea termica a energiei au devenit obstacole importante în design-ul procesoarelor moderne cu performanţe ridicate.
  „*further increasing the clock frequency is also getting more and more difficult because (i) of **heat problems** and (ii) of **too high energy consumption**. The latter is not only a technical problem for both server farms and mobile systems, but in the future, it is also going to become a **marketing weapon** targeted at the growing number of environmentally aware consumers and companies in **search of a greener computer**.*"

Environment Protection Agency (EPA): computers consume 10% of commercial electricity consumption (2008)

- This include peripherals, possibly also manufacturing
- A Department of Energy (DOE) report suggested this percentage is much lower (3.0÷3.5%)
- Internet Data center growth was cited as a contribution to the 2000/2001 California Energy Crisis

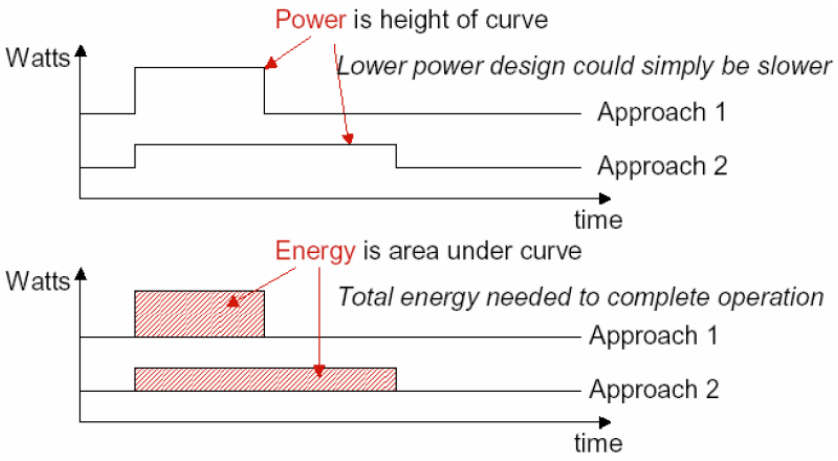
Challenges about power consumption

- Understand where and why power is dissipated



level
nuge
% of
ă de
arele

- Din ce în ce mai multe procesoare sunt integrate în dispozitive dedicate în care memoria disponibilă este limitată, cum ar fi PDA-urile şi sistemele mobile. Limitările dimensiunii memoriei rezultă din considerații cum ar fi spațiul, greutatea, consumul de putere si costul de producție.
- Cercetările au arătat că viteza circuitelor (cu implicații asupra performanței de procesare) se îmbunătățeşte cu aproximativ 2% pentru o reducere a temperaturii din cip cu 10°C [Sch02]
  (if $T_{\mu P}$ 10°C IPC 2%).
- Metrici folosite:
  - P = $[W]_{SI}$ watt – puterea medie instantanee = puterea medie disipată într-un ciclu de tact procesor
  - E = $[J]_{SI}$ joule – energia = produsul dintre putere şi timpul total de execuție: E = P * t. Această metrică este în strânsă legătură cu timpul de viață al bateriei sistemului dedicat



Sistemul de calcul		Număr nuclee per cip	Frecvența de procesare	Puterea medie instantanee
Server de ultimă generație				~130÷170W
Desktop de ultimă generație[5]				~70÷100W
Laptop de ultimă generație				~30W
Laptop cu baterie optimizată				~3÷10W
Procesoare dedicate				~0.5÷1W
Procesoare de semnal (DSP)				~100mW
Microcontrollere				~10mW
Procesoare multicore	AMD Opteron X4 (Barcelona)	4	2.5 GHz	120W
	Intel Nehalem	4	2.5 GHz	100W
	IBM Power 6	2	4.7 GHz	100W
	Sun Ultra SPARC T2 (Niagara 2)	8	1.4 GHz	94W

Tabelul 1. Consumul de putere pentru fiecare tip de sistem de calcul

[5] Pentium P4 Extreme Edition consumă 135W

Puterea dinamică reprezintă principala sursă de putere disipată în microprocesoarele realizate în tehnologie CMOS şi este definită astfel:

$$P_d = C \cdot V_{dd}^2 \cdot a \cdot f \text{ ,}$$

Unde,

- C este capacitatea circuitului ($[nF]_{SI}$). Depinde de lungimea firelor, dimensiunea tehnologiei.
- Vdd (*tensiunea de alimentare – 2.2V*) şi f (*frecvenţa procesorului*) sunt determinate de procesul tehnologic cu toate că multe decizii de proiectare microarhitecturală conduc la diminuarea frecvenţei[6].
- a – *factorul de activitate* (numărul mediu de scrieri – sau numărul de tranziţii din 0 în 1 şi invers) aferent fiecărei componente microarhitecturale.

$$P_{Mean} = \frac{\int_0^T P(t) \cdot dt}{T}$$

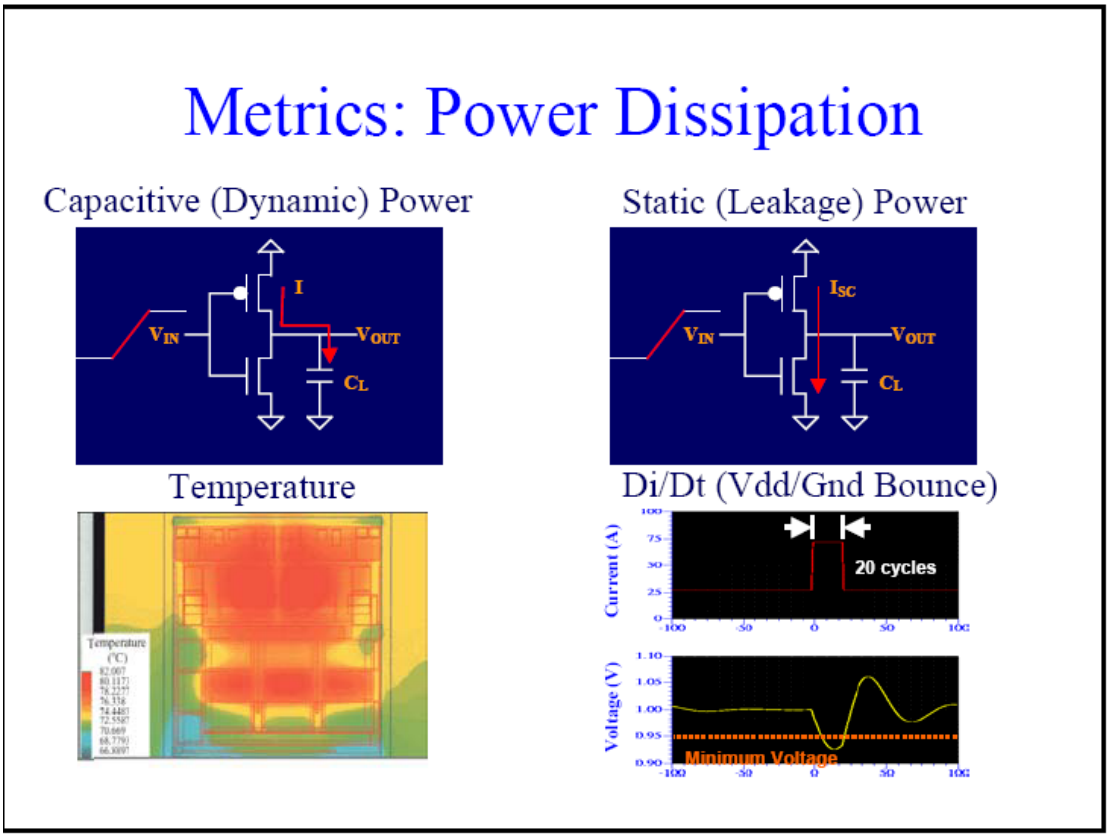
$$E = P_{Mean} \cdot T$$

Unde T reprezintă timpul total de simulare / execuţie măsurat în perioade de tact iar P este puterea dinamică (P_d)

Puterea statică începe să aibă un aport tot mai mare la puterea totală consumată.

Puterea statică este puterea consumată într-un sistem în absenţa oricărei comutaţii şi este definită ca produsul dintre tensiunea de alimentare şi curentul static. Această pierdere este datorată dispozitivelor conectate la condensatori, cum ar fi tranzistori sau diode care, chiar dacă sunt închise conduc o cantitate mică de curent. Chiar dacă acest current este foarte mic el poate descărca lent condensatorul. **Imperfecţiunile nedorite ale unor materialele dielectrice folosite în condensatori pot contribui la puterea statică consumată**. Materialul dielectric nefiind un izolator perfect permite circularea unui curent static (de pierdere) mic care va descărca lent condensatorul. Contribuţia puterii statice la totalul de putere consumată intr-un sistem este preconizată să crească in tehnologiile viitoare datorită cresterii exponenţiale a curentului static pierdut odată cu scalarea tehnologiei. International Technology Roadmap for Semiconductors (ITRS – 2006) [http://public.itrs.net/] **anticipează că puterea statică poate ajunge si la 50% din totalul de putere în următoarea generaţie de procesoare**.

[6] La anumite procesoare o frecvență ridicată făcea imposibilă predicția următoarei instrucțiuni într-o perioadă de tact



Temperatura ajunge până la **83°C** (*se pot găti ochiuri liniştit*). Cipul are ≈ 2.5x2.5 cm^2
Inductorii nu au o rezistenţă stabilă în timp precum conductorii.
Variaţia tensiunii instantanee (Vdd/Gnd Bounce – calea directă a curentului între VDD şi GND când ambii tranzistori NMOS şi PMOS conduc) determină pentru cei 20 de ciclii de $tact_{CPU}$ apariţia unui curent de scurgere.

$$v = L\frac{di}{dt}$$

Where,

v = Instantaneous voltage across the inductor

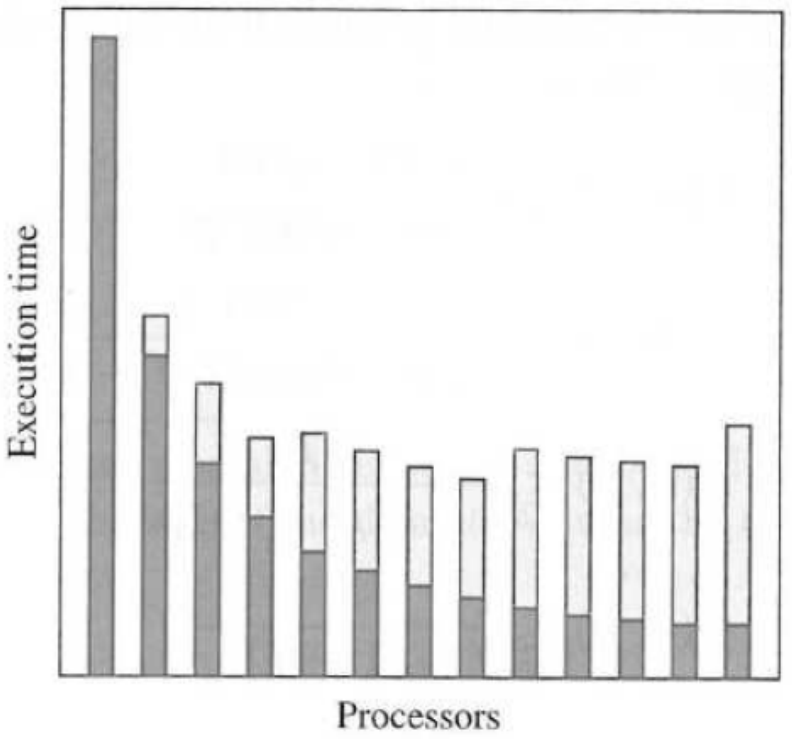
L = Inductance in Henrys

$\frac{di}{dt}$ = Instantaneous rate of current change (amps per second)

Documente utile:
1 - Power: Major Issue in Computer Architecture
2 - Power Management Features in Intel Processors

LEGEA LUI AMDAHL (Legea scalabilităţii totale)

An example of time measurements



Dark grey: time spent on computation, decreasing with p
White: time spent on communication, increasing with p

Objectives of the lecture

- How to roughly predict computing time as function of p?
- How to analyze parallel execution times?
- Understand the limit of using more processors

Computational Science and Engineering

Old School	New School
• Increasing clock frequency is primary method of performance improvement	• Don't count on frequency increases as main driver of your performance improvement
• Don't bother parallelizing an application, parallel computing is odd and expensive	• Nobody builds one core processors anymore. Processors' parallelism is the de-facto method for performance improvement.
• Less than linear scaling for a multiprocessor is failure	• Given the switch to parallel hardware, even sub-linear speedups are beneficial as long as you beat the sequential

Notation

n	problem size
p	number of processors
$\sigma(n)$	inherently sequential computaiton
$\varphi(n)$	parallelizable computation
$\kappa(n,p)$	parallelization overhead

$$\text{Speedup } \Psi(n,p) = \frac{\text{Sequential execution time}}{\text{Parallel execution time}}$$

$$\text{Efficiency } \varepsilon(n,p) = \frac{\text{Sequential execution time}}{\text{Processors used} \times \text{Parallel execution time}}$$

Ideal ar fi ca un SMM dotat cu N procesoare să proceseze un program de N ori mai rapid decât un sistem monoprocesor, cerinţă numită **scalabilitate** completă. În realitate, acest deziderat de scalabilitate nu se realizează din multiple motive. În privinţa **scalabilităţii, aceasta este mai uşor de realizat pe un sistem cu resurse distribuite decât pe unul având resurse centralizate,** întrucât acestea din urmă constituie un factor de "strangulare" a activităţii.

Dintre **cauzele care stau în calea unei scalabilităţi ideale**, şi care generează limitări semnificative (neliniare) de performanţă, se amintesc:

1. **Gradul de secvenţialitate intrinsec al algoritmului executat**. Aşa de exemplu, există în cadrul unui algoritm operaţii secvenţiale dependente de date şi deci imposibil de partajat în vederea procesării lor paralele pe mai multe procesoare.

$$\left.\begin{array}{l} n = y + z; \\ a = n + b; \end{array}\right\} \text{scalar secvential dependent de RAW}$$

100% = secvenţial

$$\left.\begin{array}{l} \text{for } i = 1 \text{ to } 10 \\ \quad A(i) = B(i) + C(i); \end{array}\right\} \text{paralelizabil pe 10 procesoare}$$

(1-f)×100%=paralelizabil

Accelerarea S pentru un SMM cu N procesoare este, prin definiţie:

$$S = \frac{T_s}{T_N}$$

unde:

T_s = timpul de execuţie pentru cel mai rapid algoritm secvenţial care rezolvă problema pe un monoprocesor (SISD)

T_N = timpul de execuţie al algoritmului paralel executat pe un SMM cu N μprocesoare.

Dacă notăm cu f = fracţia (procentajul) din algoritm care are un caracter eminamente secvenţial, f∈[0,1], putem scrie:

$$T_N = f \cdot T_s + \frac{(1-f) \cdot T_s}{N},$$

adică

$$S = \frac{T_s}{f \cdot T_s + \frac{(1-f) \cdot T_s}{N}}$$

sau:

$$S = \frac{1}{f + \frac{(1-f)}{N}}$$

Legea lui G. Amdahl, 1≤S≤N (scalabil)

Legea lui G. Amdahl sugerează că un procentaj (fx100%) oricât de scăzut de calcule secvenţiale impune o limită superioară a accelerării (1/f) care poate fi obţinută pentru un anumit algoritm paralel pe un SMM, indiferent de numărul N al procesoarelor din sistem şi topologia de interconectare a acestora.

2. **Timpul consumat cu sincronizarea şi comunicarea între procesele** (firele de execuţie) rezidente pe diversele (**N**) procesoare din sistem.
3. **Imposibilitatea** sau neputinţa **balansării optimale** – de către sistemul de operare şi sistemul hardware – a activităţii procesoarelor din sistem (**Load Balancing**). Frecvent, nu se poate evita situaţia în care anumite procesoare să fie practic inactive sau cu un grad scăzut de utilizare.
4. **Planificarea sub-optimală**, din punct de vedere al performanţei globale, **a proceselor din punct de vedere software** (activare proces, punere în aşteptare a unui proces, schimbarea contextului în comutare a proceselor etc.)
5. **Operaţiile de I/O**, în cazul nesuprapunerii lor peste activitatea de execuţie a task-ului de către procesor (overlapping).

Se presupune, spre ex., ca **f=10%** din timp un program nu poate fi accelerat prin paralelizare pe un sistem cu **N=100 de procesoare**. Se consideră că pentru a rula această parte secvențială de două ori mai rapid decat pe un procesor simplu, este nevoie de un procesor complex (Out-of-Order) care are de 10 ori mai multe resurse (complexitate) decat procesorul simplu. Aplicand legea lui *Amdahl*, accelerarea obtinuta pe un **sistem omogen cu 100 de nuclee simple** este:

$$S=1/[f+(1-f)/N]=1/[0,1+0,9/100]=\mathbf{9.17}$$

Daca utilizam N=90 procesoare omogene si un procesor OoO (echivalentul a zece procesoare omogene, și care rulează partea secventiala de doua ori mai rapid decat pe un procesor simplu) vom obtine urmatoarea accelerare (pe **sistem eterogen**):

$$S=1/[0,1/2+0,9/90]=\mathbf{16.67}$$

Totusi, superioritatea sistemelor eterogene fata de sistemele omogene va trebui dovedita mult mai convingator prin simulari complexe, atat pe *benchmark*-uri secventiuale cat si paralelizate (IPC, energie consumata, buget tranzistori, arie integrare etc.). De altfel, chiar si actualele multiprocesoare comerciale sunt neomogene, avand un nucleu superscalar *out of order* foarte puternic (IBM Cell BEA, Intel IXP – procesoare de retea etc.)

Dintre **metricile succesului** in cadrul **dezvoltarii sistemelor *multicore*** se amintesc:

- **productivitatea programării**
- **performanţa aplicaţiei.**
  - minimizarea acceselor la datele aflate în afara memoriilor locale
  - optimizarea balansării încărcării procesoarelor
  - optimizarea comunicaţiilor şi sincronizărilor

Q1: If we know that 90% of the computation can be parallelized, what is the maximum speedup we can expect from using 8 processors?
Q2: If 18% of the operations in a parallel program must be performed sequentially, what is the maximum speedup achievable?
Q3: Assume 0.2% of the runtime of a program is not parallelizable. This program is supposed to run on the supercomputer which consists of 300 cores. Under the assumption that the program runs at the same speed on all of those cores, and there are no additional overheads, what is the Efficiency ?

SIMULATOARE SOFTWARE DEDICATE MICROARHITECTURILOR DE CALCUL.

Simulatorul. Ce este? Este necesară simularea?

- ❑ Istoria procesoarelor contrapune două paradigme pentru creşterea performanţei, bazate pe software şi respectiv pe hardware.

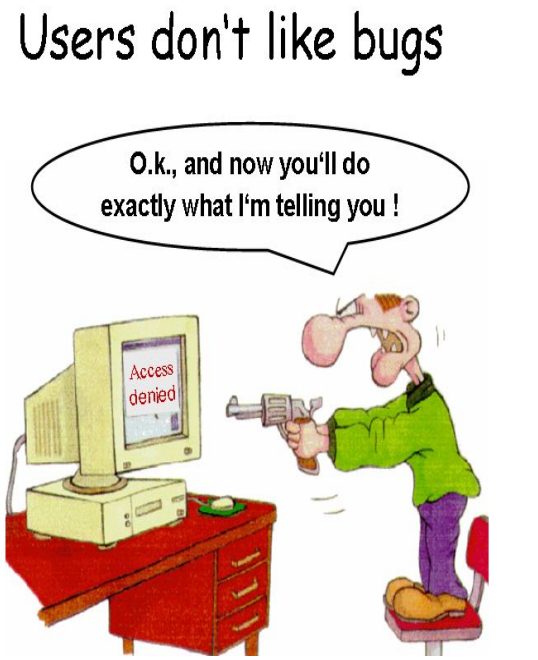
- ❑ În procesul de proiectare al procesoarelor, aferent generaţiilor viitoare, accentul principal nu se mai pune pe implementarea hardware, ci pe proiectarea arhitecturii în strânsă legătură cu aplicaţiile potenţiale. Se porneşte de la o arhitectură de bază (generică), puternic parametrizată, care este modificată şi îmbunătăţită dinamic, prin simulări laborioase pe programe de test (benchmark-uri) reprezentative.

- ❑ **Simulator dedicat unei arhitecturi de calcul** – instrument software (aplicaţie/program) utilizat în exploatarea / cercetarea / şi îmbunătăţirea performanţelor unei microarhitecturi. De obicei funcţionarea microarhitecturii se simulează la nivel de ciclu maşină permiţând vizualizarea tuturor resurselor hardware la finele fiecărui ciclu.

Provocari majore legate de metodologia de simulare si tehnologia software folosita in dezvoltarea simulatoarelor

- **Proiectarea si implementarea software**
  - Interfata cu utilizatorul si crearea resurselor.
  - Nucleul functional al simulatorului.
  - Metodologia de simulare. Benchmark-uri. Optimizatoare de cod obiect (*schedulere* statice).
  - Distribuirea simularii pe mai multe resurse hardware (arhitectura client – server in retea) sau simularea pe o singura statie serial cate o configuratie microarhitecturala una dupa alta.
- **Modelarea capabilitatilor de simulare in functie de metricile urmarite**
  - Nucleul de procesare al microarhitecturii (unitati executie, statii rezervare, structura pipeline)
  - Nivelul ierarhic de memorie: cache-uri de instructiuni / date – L1, L2 sau chiar L3, DRAM
  - Structuri de predictie implementate
  - Arhitecturi ILP, TLP, monocore / multicore
- **Reducerea timpului de executie aferent unei simulari**
  - *Sample simulation* – gasirea unui nucleu de instructiuni care aproximeaza executia intregului benchmark
- **Implementarea unui Automatic Design Space Exploration**
  - Metode de cercetare euristica in spatiul tuturor parametrilor arhitecturii, compilatorului si al aplicatiilor pentru gasirea optimului din punct de vedere multicriterial (performanta de procesare, putere consumata, temperatura disipata, numar nuclee, etc.)
  - Algoritmi de cautare locali (in vecinatatea unei configuratii microarhitecturale) de tip *hill climbing*, *simulated annealing*
  - Metode avansate de invatare automata (*machine learning*), algoritmi genetici, algoritmi stigmergici bazati pe miscarea coloniilor de furnici, etc.

❑ **Simularea este necesară pentru că:**



Dupa mai bine de **douazeci de ani de preocupari in proiectarea si exploatarea arhitecturilor de calcul cercetatorii** au ajuns la concluzia ca **simulatoarele au devenit instrumente primordiale in cercetarea si dezvoltarea din domeniul stiintei calculatoarelor**. Principalele **avantaje** ale simulatoarelor comparativ cu procesoarele reale sunt: **costul de implementare si timpul de dezvoltare mai mic**, **flexibilitatea si extensibilitatea** care permit arhitectului de microprocesoare sa **evalueze mai rapid performanta unei plaje largi de arhitecturi** si sa cuantifice eficienta fiecarei imbunatatiri (inovatii microarhitecturale sau optimizari la nivelul

compilatorului). Cu toate acestea, **timpul indelungat de simulare** (zeci de ore pentru doar o singura configuratie arhitecturala) si **acuratetea scazuta a rezultatelor simularii** (sau imposibilitatea cautarii, analizei si simularii complete in spatiul enorm al tuturor parametrilor arhitecturii, compilatorului si al aplicatiilor[7]) **limiteaza eficienta simulatoarelor**. Probabil cel mai important impediment întâlnit se datorează constrângerilor temporale; optimizarea unei arhitecturi dovedindu-se un proces lent care necesită putere de calcul considerabilă. Pe langa importanta dovedita in domeniul cercetarii stiintei calculatoarelor in general si al arhitecturilor de calcul in special, in ultimul timp, **simulatoarele au devenit instrumente pedagogice valoroase** care permit studentilor să modifice **interactiv** parametrii configuraţiilor hardware şi să observe efectele lor într-o manieră mult mai accesibilă, sa **vizualizeze componentele microarhitecturale** si **interactiunea** dintre acestea, sa inteleaga si sa utilizeze mai bine principiile si conceptele teoretice din domeniu.

Joshua J. Yi, David J. Lilja – *Simulation of Computer Architectures: Simulators, Benchmarks, Methodologies, and Recommendations*, IEEE TRANSACTIONS ON COMPUTERS, VOL. 55, NO. 3, MARCH 2006.

METODOLOGIA DE SIMULARE poate fi de două (trei, dar a treia reprezintă un **hibrid** între cele două fundamentale) tipuri:

Execution driven simulation

- ⇨ caracterizată de cunoaşterea în fiecare moment (ciclu "pipe") a conţinutului resurselor arhitecturale (regiştri, locaţii de memorie, unităţi funcţionale).
- ⇨ Simularea se face foarte detaliat, la nivel de ciclu de execuţie al procesorului.
- ⇨ **Outputs**: - conţinutul resurselor, gradul de încărcare al acestora, rate de procesare, de hit etc.
  - Fişiere **trace** care conţin toate instrucţiunile maşină ale programelor de test în ordinea în care se execută.

Trace driven simulation

- ⇨ analizează secvenţial toate instrucţiunile din trace-urile generate de simulatorul bazat pe execution driven, cu scopul de a determina instanţa optimală a arhitecturii - *procesorul ce urmează a fi implementat în hardware*. TDS se pretează la *simularea cache-urilor* de date şi instrucţiuni, mecanismelor de memorie virtuală etc., datorită faptului că oferă pattern-uri reale de adrese, în urma execuţiei unor programe reprezentative.

[7] Presupunem că simulăm o microarhitectură cu **10 parametrii (FR, IBS, IRmax, SizeIC, SizeDC, latentaDRAM, ROBsize, RenBuffersize, optimizări_compilator**), fiecare parametru putandu-se afla in **4** stari diferite $4^{10}=2^{20}=1048576$ configuraţii distincte de simulat * 24h de simulare per configuraţie pe un sistem monoprocesor **2872 de ani** (Se reaminteşte că pe n digiţi se pot reprezenta în baza b un b^n numere: de la $00\ldots.00_b$) până la $(b-1)(b-1)\ldots.(b-1)(b-1)_b$ de la 0 la valoarea maximă $(b-1)*b^0+(b-1)*b^1+(b-1)*b^2+\ldots.+(b-1)*b^{n-1}=(b^n-1)$ deci b^n numere.

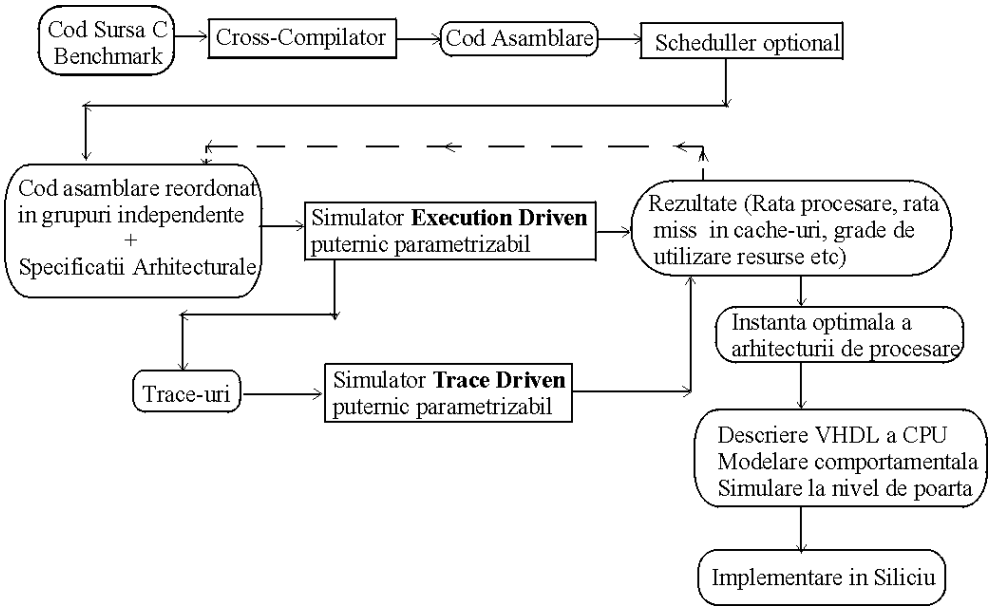


Figura 1. Etapele de simulare, comparare şi determinare efectivă ale unei arhitecturi optime, pornind de la sursa HLL (High Level Languages) a programelor de test şi până la implementarea hardware a arhitecturii

Simulatoarele determina: *performanta de procesare + model de putere* (**CACTI, WATCH**) *estimare putere /energie consumata + configuratie termica && configuratie microarhitecturala spatiala* (**Quilt, HotSpot**) *hotspots identificare DTM* (manager termal dinamic).

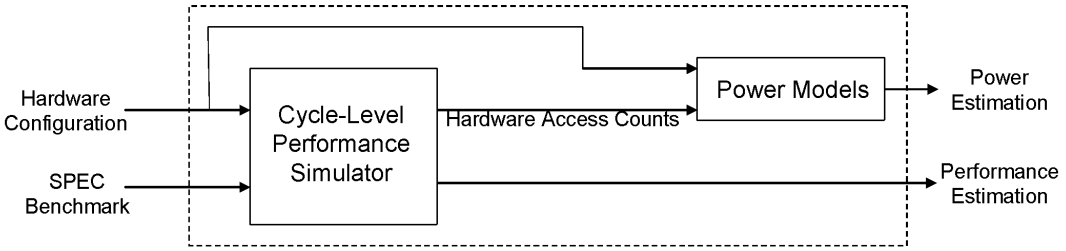


Figure 2. The structure of the power consumption simulator

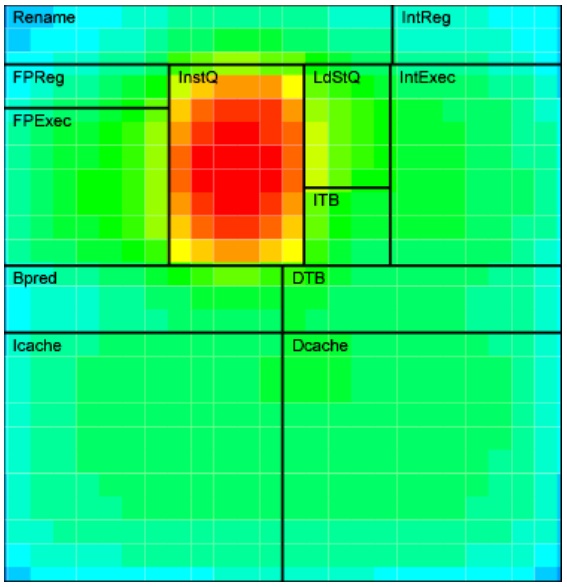


Figure 0.3. The thermal map of the Alpha 21364 architecure

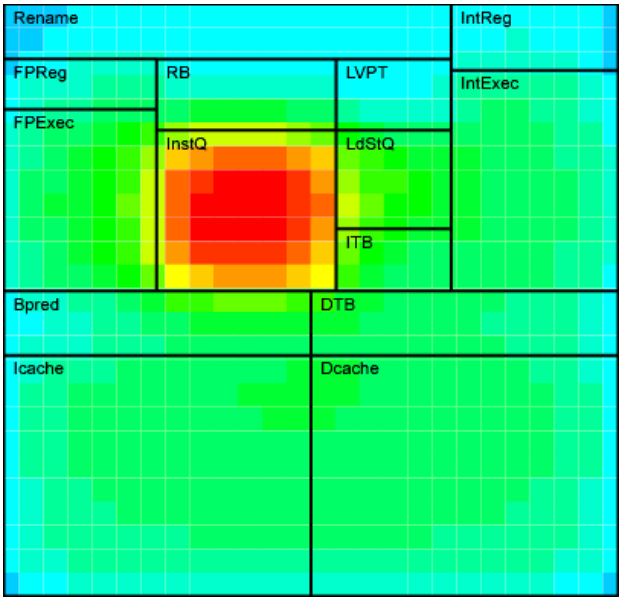


Figure 0.4. The thermal map of the Alpha 21364 architecture enhanced with RB and LVPT

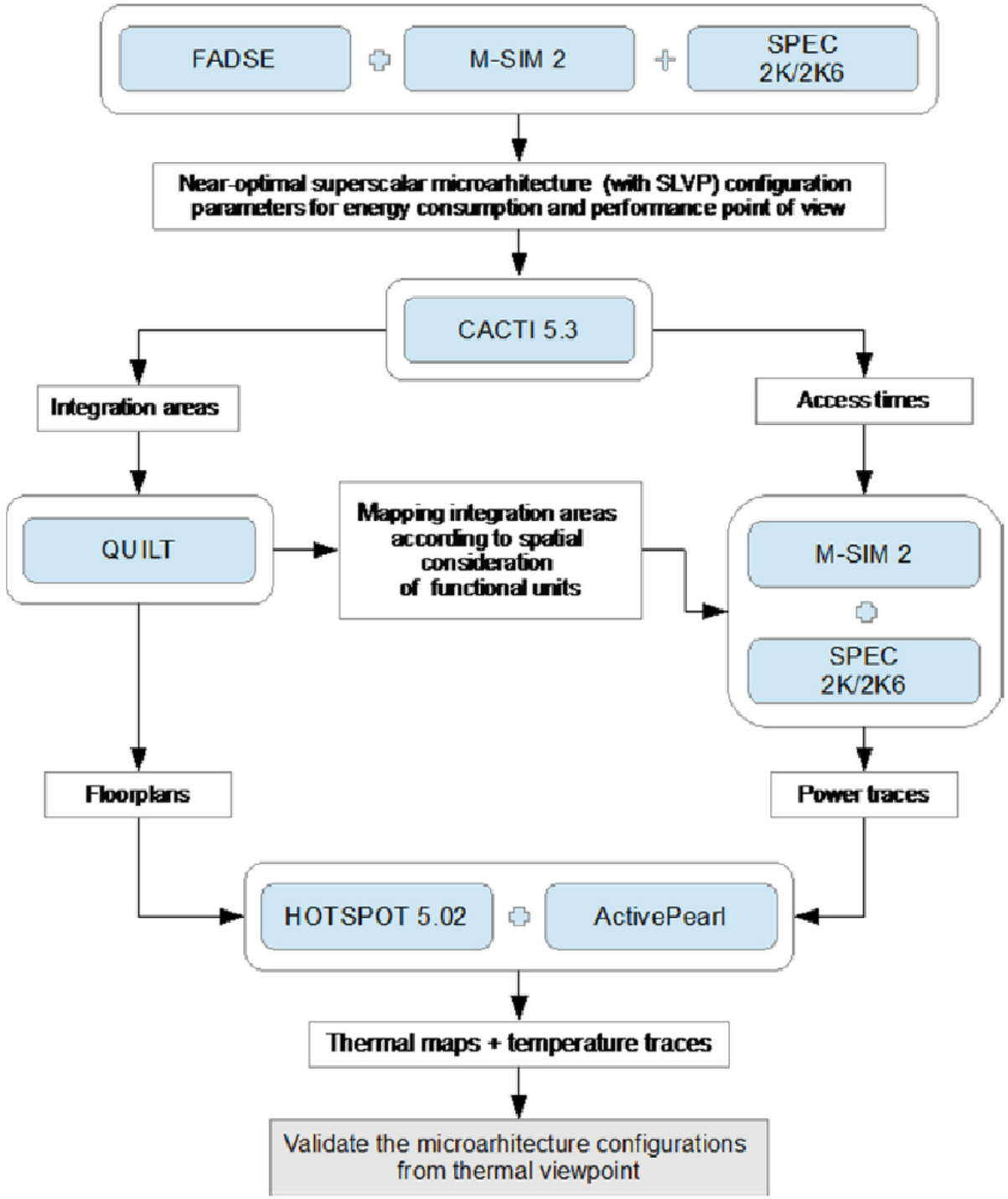


Fig. 2. Workflow for finding a near optimal microarchitecture from the thermal viewpoint.

Din perspective tehnologiei software de implementare a simulatoarelor se disting platforme:

monolitice *(SimpleScalar).*

Avantaje: disponibilitate gratuita, flexibilitate, portabilitate, extensibilitate
Dezavantaje: dificultatea depanarii si a reutilizarii codului la dezvoltarea unei arhitecturi noi

Execution driven (cycle level**) – *monocore*:**

- Simulatorul de cache-uri (*sim-cache*), de profil (*sim-profile*)
- Predictorul de salturi (*sim-bpred*)
- Simulatorul superspeculativ de complexitate ridicată out of order (*sim-outorder*)
- Simulatorul *TLP* (cu paralelism simultan la nivelul firelor de executie – *msim*)

Trace driven (trace level):
- Simulatorul de (*hotspot*)

Execution driven (cycle level**) – *multicore***
- Simulatorul *TLP* (cu paralelism simultan la nivelul firelor de executie – *multi2sim, msim-cmp*)

modulare (***UniSim***).

Avantaje:
- permite simularea multicore-urilor
- mediu de simulare **modular** (se pot dezvolta separat, creşte încrederea în implementare – module mai mici, uşor de testat)
- interfeţe de **comunicare standardizate intre module**
- metodologii diferite de simulare: **Cycle level** / **Transaction level**
- posibilitatea de a introduce simulatoare existente (wrapere), de a lega alte module (gen **CACTI** pentru consum de putere)
- orientat obiect duce la **reutilizare, dezvoltare facilă** a unei arhitecturi noi

Dezavantaje: dificultatea depanarii si complexitate ridicata la arhitecturi cu numar mare de core-uri, foarte instabil, documentaţia redusa (inexistenta) a codului

Execution driven (cycle level**) – *monocore / multicore*:**
- procesor ARM cu un singur nucleu (**arm-score**)
- procesor PowerPC405 cu mai multe nuclee (**ppc-mcore**).
- procesor PowerPC405 cu un singur nucleu şi care are integrată componenta de consum de putere (**ppc-score_power_estimation**).
- procesor PowerPC405 cu suport pentru thread-uri (se pot scrie benchmark-uri care pornesc fire de execuţie, care sunt schedulate de procesor pe fiecare nucleu, pot fi sincronizări între aceste fire, se poate simula un comportament mult mai real al aplicaţiilor şi a dependenţelor care există între acestea)**/opt/public/components/CycleLevel/simulators/ppc-pthread/cmp**

Simularea la **nivel tranzactional**[8] in sistemele *multiccore / many-core*.
- Apare datorita **lipsei de productivitate din programarea paralela** (in care este necesara pastrarea atomicitatii a variabilelor partajate prin sectiuni critice – un singur procesor scrie la un moment dat o variabila)
- Modelarea la nivel de tranzacţii = separarea comunicării dintre module (hardware sau software) de implementarea lor (procesul computaţional din sistem)
- Metodologia TLM este utilă în modelarea unor arhitecturi care executa programe paralele bogate in sectiuni critice (ex.: implementarea lock-unlock, bariere) si le inlocuiesc cu alternative **checkpoint, commit, rollback** specific tranzactiilor din bazele de date.
- procesor PowerPC cu doua nuclee (**PowerPC-TLM**)

[8] Simularea la nivel de tranzactii se bazează pe limbaje de programare de nivel înalt, aşa cum este SystemC, un limbaj derivat din C++ pentru modelarea la nivel de sistem hardware-software (vezi http://www.systemc.org/) şi pune în evidenţă conceptul de separare a comunicaţiei, faţă de procesul computaţional din cadrul sistemului. În abordarea la nivel transactional (TLM), componentele sunt modelate ca şi module care pot executa o mulţime de procese concurente, care calculează şi le reprezintă comportamentul. Aceste module comunică prin mesaje sub formă de tranzacţii folosind canale de comunicaţie abstracte. Interfeţele TLM sunt implementate în cadrul canalelor de comunicaţie pentru a încapsula protocoale de comunicaţie. Pentru a putea comunica, un proces trebuie doar să acceseze aceste interfeţe, prin porturile pe care le are modulul cu care se doreşte comunicarea.

Alte instrumente software utile in analiza si proiectarea microarhitecturilor[9]
Asambloare, *link*-editoare, *debuggere*. *Cross*-compilatoare

Creşterea în performanţă şi complexitate a microprocesoarelor moderne datorate tehnicilor avansate gen *pipelining*, execuţie *out-of-order* , predicţie şi execuţie speculativă, presupune un efort suplimentar de proiectare şi verificare pentru dezvoltarea şi implementarea de produse viabile. Pentru depăşirea acestor probleme, **proiectanţii de microarhitecturi au explorat diverse modalităţi de *transfer de funcţionalitate* la nivelul compilatorului**. Începând cu procesoarele RISC VLIW şi continuând cu cele EPIC – versiunile 1 şi 2 de procesoare Intel Itanium, compilatorul a jucat un rol important în simplificarea arhitecturii la nivel hardware menţinând totodată tendinţele curente de creştere a performanţei.

Asambloare şi ***Link*-editoare:**
./configure --host=i386-*-linux --target=sslittle-na-sstrix --with-**gnu-as** --with-**gnu-ld**

[9] Pentru detalii suplimentare legate de instrumente software utile în analiza şi proiectarea microarhitecturilor a se vedea capitolul 3 din lucrarea 4 – „*Predicţia dinamică a valorilor în microprocesoarele generaţiei următoare*" din bibliografia obligatorie prezentată în cursul introductiv şi în fişa disciplinei.

Debuggere:

gcc *–c –g –O* *vpred.c

make

gdb

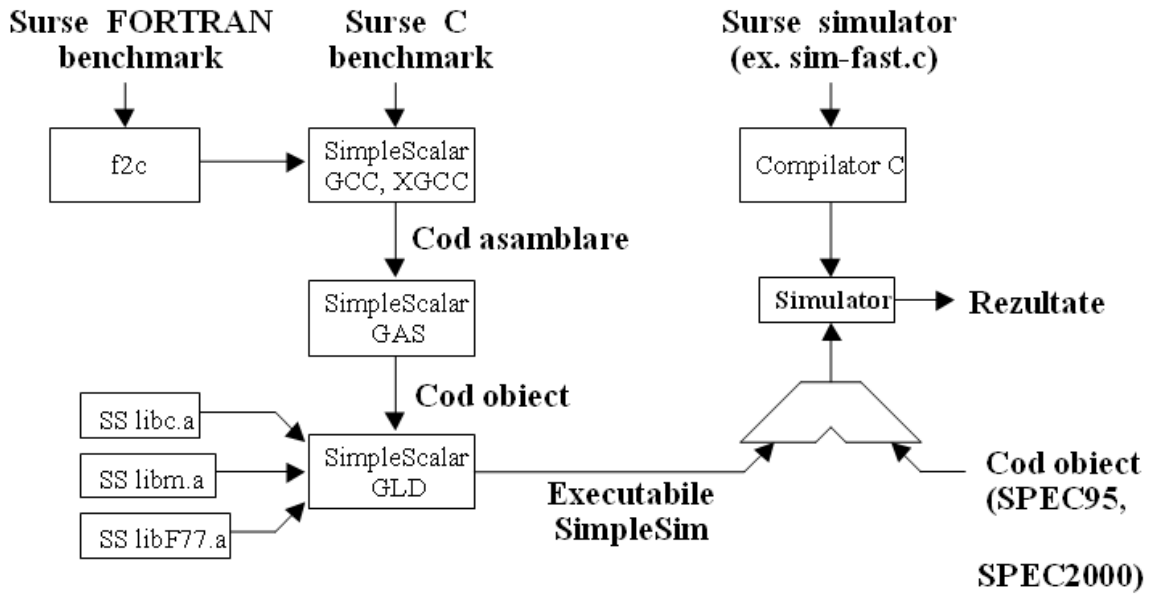
***Cross*-compilatoare**:

crosstool-**ppc405**-4.1.1.tar.gz

crosstool-**ppc750**.tar.gz

crosstool-**arm**.tar.gz

crosstool-**mips**.tar.gz



ALEGEREA PROGRAMELOR REPREZENTATIVE PENTRU DETERMINAREA PERFORMANȚEI DE PROCESARE

Cele mai bune soluții, în ordine descrescătoare, sunt:

- ideal utilizatorul doreşte **simularea aplicațiilor reale**, care sunt executate pe un sistem (**compilatoare, procesoare de text, calcul tabelar, emulatoare de sisteme de operare, baze de date, jocuri, procesare de imagine / sunet**). Aplicațiile reale au intrări / ieşiri şi diverse opţiuni de selecţie în momentul rulării. Principal dezavantaj: **lipsa portabilităţii** acestora (dependenţa de sistemul de operare şi/sau de compilator). Sporirea portabilităţii presupune modificarea codului sursă şi/sau eliminarea anumitor secţiuni de gen grafică interactivă, etc, transformând în final aplicaţia.

- Folosirea **programelor de test standardizate (*benchmark*-uri** SPEC, SYSMARK**).** Benchmark-urile SPEC sunt programe reale modificate în vederea creşterii portabilităţii şi minimizării efectului dispozitivelor I/O asupra performanţei. Un **avantaj major** al folosirii suitelor de benchmark-uri îl reprezintă faptul că "slăbiciunea" oricărui benchmark este atenuată de prezenţa celorlalte programe de test. Orice concluzie se trage de pe urma simulării întregii suite.
- Simularea unor nuclee (***kernels***) ale aplicațiilor reale (**Linpack**, **Bucle Livermore**).
  - Buclele Livermore reprezintă o colecţie de 24 structuri repetitive DO-loops implementate în Fortran. Constituie o mixtură de bucle vectorizabile şi nevectorizabile care testează capabilităţile de calcul ale sistemului hardware şi abilităţile software în compilarea eficientă a codului în vederea vectorizării buclelor de program.
  - LINPACK reprezintă o colecţie de subroutine scrise în Fortran care analizează şi rezolvă sisteme de ecuaţii liniare şi probleme de complexitate cel puţin pătratică. Nucleul LINPACK constituie o **măsură a puterii de calcul a unui sistem ce operează în virgulă mobilă**.

- **Toy Programs** – programe simple de test având până la 100 de linii de cod şi care produc rezultate apriori cunoscute de către utilizator. Exemple: *Ciurul lui Eratostene, Quicksort, Puzzle, Hanoi*, suita de benchmark-uri Stanford.

- Folosirea unor **programe sintetice** (dedicate) care **imită din punct de vedere statistic comportamentul** (frecvența operațiilor şi a operanzilor, etc) **a unui larg set de aplicații**. Sunt similare în filozofie cu benchmark-urile de tip kernel şi utilizate de către proiectanții de compilatoare. Exemple: Whetstone, Dhrystone.

BENCHMARK-URI STANDARDIZATE.

Benchmark-urile reprezintă seturi de aplicaţii dezvoltate pentru evaluarea celor mai importante metrici (performanță, consum de putere, etc.) ale sistemelor de calcul (desktop, server, embedded, rețele) şi pot fi folosite pe diverse platforme (procesor + Sistem de Operare): UNIX, Linux şi Microsoft Windows (dezvoltate de Microsoft).

Benchmark-urile **SPEC (Standard Performance and Evaluation Corporation**) au fost dezvoltate de "*SPEC's Open Systems Group*" (OSG), care înglobează peste 30 de producători de calculatoare, integratori de sisteme, autori si consultanţi din întreaga lume.

BAPCo (Business Applications Performance Corporation) reprezintă un consorţiu non-profit având ca scop principal dezvoltarea şi distribuţia unui set de benchmark-uri performante bazate pe aplicaţii de uz general dar şi pe sisteme de operare standardizate folosite în industrie. Rolul benchmark-urilor BAPCo este de **a evalua tehnologiile implementate pe diferite platforme**. Din BAPCo fac parte: AMD, Apple, ARCintuition, CNET, Dell, Hewlett-Packard, Intel, Lenovo, Microsoft, NVIDIA, Sony, Toshiba, VIA Technologies, ZDNet and Ziff Davis Media.

TPC (Transaction Processing Council) a fost creat prin mijlocul anilor 1980 de către un grup de ingineri producători de calculatoare, integratori de sisteme. Scopul lor era de a crea un set de **benchmark-uri realiste pentru procesarea tranzacţiilor** (măsoară abilitatea sistemului de a manipula tranzacţii, constând în principiu în **accese / interogări, actualizări ale bazelor de date**). Performanța este raportată în tranzacţii per secundă.

SYSmark 2007 reprezintă cea mai recentă versiune de benchmark-uri BAPCo. Reflectă patern-uri de aplicaţii (*Video creation, E-learning, 3D Modeling, Office Productivity*) ale utilizatorilor (clienţi) din domeniul economic. Caracterizează performanța economică a unui client (aplicaţii software dedicate sectorului economic pe platformă Microsoft Windows Vista).

SPEC CPU2006 este a 5-a versiune majoră a seturilor de benchmark-uri SPEC CPU, care, în 1989 a devenit primul standard acceptat la scară largă pentru compararea performanţelor la calcul intensiv pe o varietate de arhitecturi.

"*Tehnologia sistemelor de calcul se dezvoltă aşa de repede, încât trebuie să oferim noi pachete de benchmark-uri, pentru a asigura un mediu de testare adecvat. Benchmark-urile standardizate, trebuie să reflecte îmbunătăţirile tehnologice aferente microprocesoarelor, noi compilatoare, aplicaţii multimedia şi transmisii de semnal audio/video/GSM, care s-a făcut în ultimii 6 ani.*"(adaptare după Kaivalya M. Dixit, președinte SPEC 1990÷2002).

CLASIFICAREA BENCHMARK-URILOR. SUITE REPREZENTATIVE

- Benchmark-uri **desktop**

- **CPU intensive** (SPEC CPU2006 – CINT2006, CFP2006; SPEC 2017 - https://www.spec.org/cpu2017/Docs/overview.html#benchmarks)
- **grafic intensive** (cu toate că multe aplicaţii grafice includ calcul intensiv şi din partea CPU) – SPECviewperf (benchmark-uri care se bazează pe biblioteci OpenGL), SPECapc (Pro/Engineer, SolidWorks 99, Unigraphics V15) – aplicaţii care folosesc din abundenţă primitive grafice.

o Benchmark-uri **server** (TPC-C, TPC-H, TPC-R, TPC-W, SPECjbb)**.**

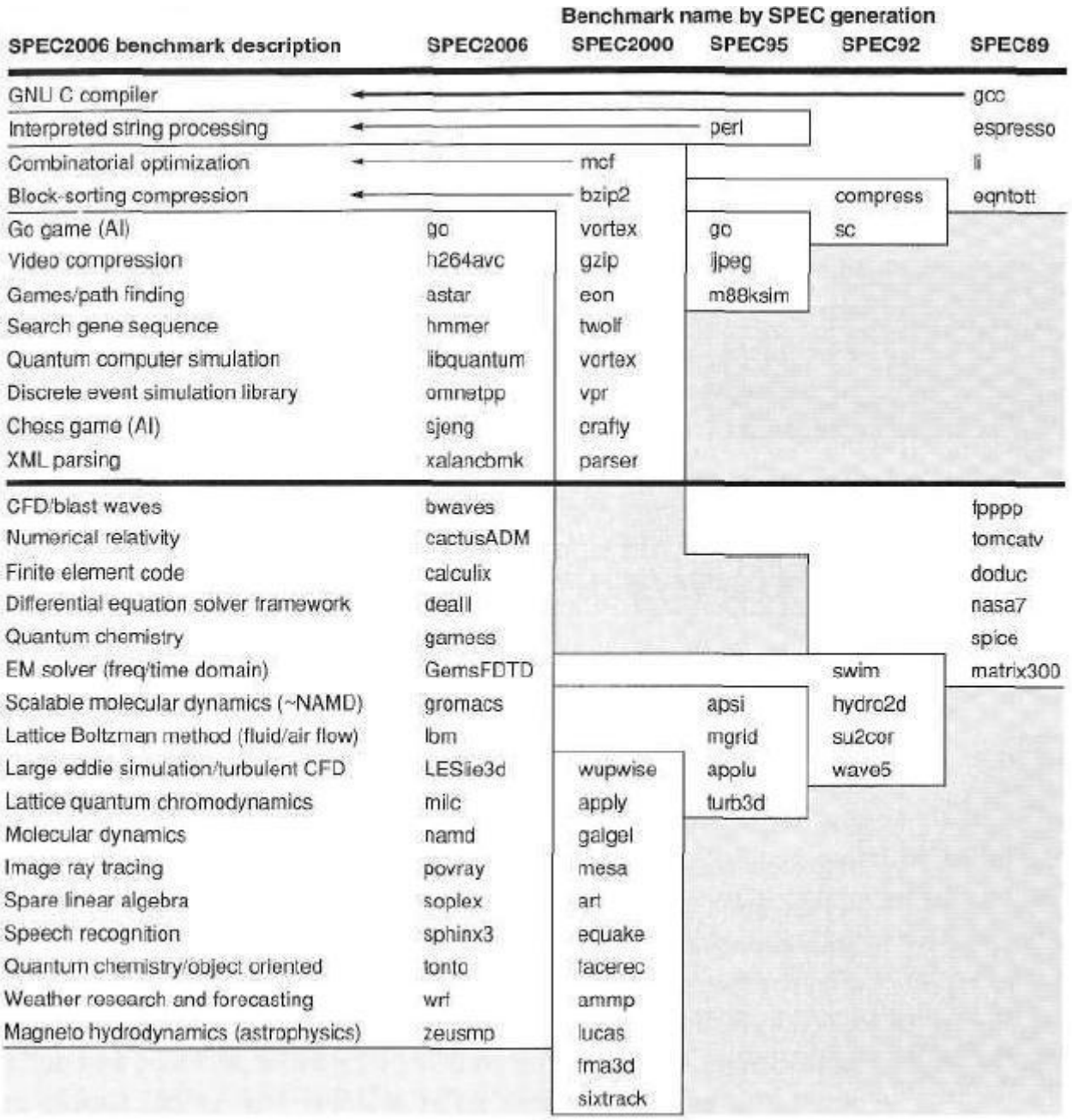
o Benchmark-uri **dedicate (*embedded*) –** EEMBC ("embassy"), SPECjvm2008, SoftFloat, MediaBench

TPC-C simulează un **mediu complex de interogări**. Se bazează pe servere de baze de date standardizate: Oracle, Microsoft SQL.
TPC-H modelează un sistem de decizii ad-hoc în care nu se ţine cont de patern-ul anterior de interogări (neexistând legături între acestea) pentru optimizarea interogărilor viitoare; **timpul de execuţie al anumitor interogări poate fi foarte mare = dezavantaj**.
TPC-R simulează un sistem de decizii pentru afaceri în care utilizatorii rulează un set standard de interogări. În TPC-R, se ţine cont de informaţiile privind interogările anteriore pentru **optimizarea** interogărilor viitoare.
TPC-W simulează activităţile unui **server web** dedicat tranzacţiilor unei companii din domeniul economic.

SPEC CPU2006 cuprinde 2 seturi de benchmark-uri: CINT2006 pentru măsurarea performanţelor în cazul calculului intensiv cu numere întregi şi CFP2006 pentru performanţele în cazul calculului intensiv în virgulă flotantă. Cele 2 seturi măsoară performanţele procesorului, arhitecturii memoriei şi compilatorului unui calculator.

Setul CINT2006 cuprinde 12 benchmark-uri bazate pe aplicaţii, scrise în limbajele C (9) şi C++ (3), iar CFP2000 cuprinde 17 benchmark-uri care realizează operaţii în virgulă mobilă, scrise în FORTRAN (6), C++ (4), C (3) şi 4 într-un "amestec" de C şi Fortran.

În ultima decadă, substanţiale îmbunătăţiri au avut loc în tehnologia compilatoarelor pentru extragerea şi valorificarea paralelismului la nivelul instrucţiunilor (ILP). Aceste aplicaţii nu cuprind toate elementele esenţiale ale aplicaţiilor multimedia şi de comunicaţie.



SPEC2006 programs and the evolution of the SPEC benchmarks over time, with integer programs above the line and floating-point programs below the line. The figure shows all 70 of the programs in the 1989,1992,1995,2000, and 2006 releases. Only 3 integer programs and 3 floating-point programs survived three or more generations.

Note that **all the floating-point programs are new for SPEC2006**. Although a few are carried over from generation to generation, the version of the program changes and either the input or the size of the benchmark is often changed to increase its running time and to avoid perturbation in measurement or domination of the execution time by some factor other than CPU time.

Benchmark Type	# of this type	Example benchmarks
Automotive/industrial	16	6 microbenchmarks (arithmetic operations, pointer chasing, memory performance, matrix arithmetic, table lookup, bit manipulation), 5 automobile control benchmarks, and 5 filter or FFT benchmarks.
Consumer	5	5 multimedia benchmarks (JPEG compress/decompress, filtering, and RGB conversions).
Networking	3	Shortest path calculation, IP routing, and packet flow operations.
Office automation	4	Graphics and text benchmarks (Bezier curve calculation, dithering, image rotation, text processing).
Telecommunications	6	Filtering and DSP benchmarks (autocorrelation, FFT, decoder, and encoder)

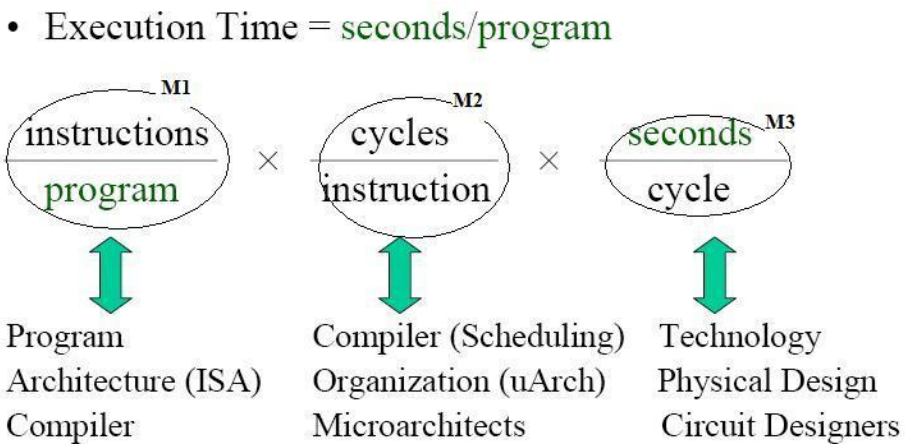
ECUAŢIA DE STABILIRE A PERFORMANŢEI PROCESORULUI (CPU)

Performanţa unui program (executat de un procesor) depinde de o combinaţie de factori:

- **Eficienţa algoritmilor utilizaţi în program** (optimizaţi / neoptimizaţi – din punct de vedere al referinţelor la memorie şi din punct de vedere al timpului de execuţie)
- **Componente software** utilizate pentru a crea şi translata codul sursă al programului în instrucţiuni maşină (**compilatoare**, translatoare, reorganizatoare de cod - scheduler[10], asambloare, etc)
- Capacitatea **sistemului de calcul de a executa eficient aceste instrucţiuni**, care pot include operaţii de intrare / ieşire (I/O) – mari consumatoare de timp.

Componenta hardware sau software a sistemului de calcul	Modul în care aceasta afectează performanţa
Algoritmul (sortare, căutare binară, folosirea tehnicilor de divide et impera, etc)	Determină numărul de enunţuri / instrucţiuni / comparaţii la nivel de cod sursă precum şi numărul de operaţii de intrare / ieşire executate.
Limbajul de programare, Compilatorul şi Arhitectura Setului de Instrucţiuni	Determină numărul de instrucţiuni asamblare / maşină corespunzătoare fiecărui enunţ de la nivel de cod sursă.
Interfaţa dintre procesor şi sistemul ierarhic de memorie	Stabileşte cât de repede vor fi executate instrucţiunile maşină.
Dispozitivele periferice (I/O) şi interfaţarea operaţiilor de intrare / ieşire cu Sistemul de Operare prin întreruperi software	Determină cât de repede vor fi executate operaţiile de intrare / ieşire.

[10] Tehnici de reorganizare a codului de tip: **loop unrolling**, **software pipelining**, **mov merging**, **mov reabsorption**, **immediate merging**, **loop interchange**, **function inlining**, **merging array**, etc contribuie la optimizarea programelor în vederea creşterii paralelismului la nivelul instrucţiunilor.



- Prima metrică (**M1 – numărul de instrucţiuni din program**) depinde de **program**, de limbajul în care este implementată aplicaţia, de **ISA procesorului** (programul poate fi scris în limbaj de asamblare) în final el fiind translatat în cod maşină şi executat, şi evident de **compilator**.
- A doua metrică (**M2 – numărul de ciclii în care se execută o instrucţiune**) poate depinde de **scheduler** (componentă a compilatorului) dacă se încearcă optimizarea şi execuţia în paralel a mai multor instrucţiuni independente, dar depinde în mod cert de **microarhitectură** (adâncimea structurii pipeline, gradul de superscalaritate[11]; proiectarea anumitor instrucţiuni flotante – înmulţire/împărţire/radical – poate fi optimizată[12]).
- Dacă primele două metrici erau **arhitecturale**, a treia (**M3 – numărul de ciclii de tact procesor dintr-o secundă**) este **tehnologică** şi depinde de fizicieni, electronişti, proiectanţi de circuite logice / secvenţiale în vederea creşterii frecvenţei de comutaţie a tranzistorilor şi de minimizare a timpului de execuţie a fiecărei faze de procesare (… decodificare / execuţie … etc.) şi a perioadei de tact a procesorului.

ŞIRETLICURI ARHITECTURALE UZUALE

- Lungimea programului este constantă.
  - Se simulează în mod repetat aceleaşi benchmark-uri, folosindu-se de acelaşi compilator
  - Uneori pot fi variate opţiuni de compilare pentru observarea impactului asupra performanţei a tehnicilor arhitecturale propuse şi în ce măsură câştigul se datorează compilatorului.
- Perioada de tact este constantă (secundă / ciclii de tact$_{CPU}$).

Asftel, în majoritatea colectivelor de cercetare academică, prin simulare execution-driven se determină performanţa de procesare prin focalizarea pe studiul **ciclilor / instrucţiune**. În consecinţă, se modifică configuraţia microarhitecturală pentru calculul IPC (instrucţiuni / ciclu de tact$_{CPU}$) optim.

[11] Efectul **hazardurilor de ramificație** (predicția greșită a instrucțiunilor de salt) este senzitiv la microarhitectură, afectând performanța de procesare. O structură pipeline cu 20 de faze de procesare cum are Intel Pentium 4 și un factor de superscalaritate de 4, sau chiar 8 instrucțiuni, determină ca în execuție să se găsească un număr de instrucțiuni de ordinul sutelor chiar miilor. Având în vedere că uneori pot trece și 100 de T_{CPU} până când se soluționează o instrucțiune de salt, rezultă necesitatea evacuării tuturor instrucțiunilor procesate pe calea greșită și reluarea de pe calea corectă, cu implicații negative asupra performanței de procesare și a energiei disipate.

[12] Dezvoltarea unor structuri de **reutilizare a rezultatelor instrucțiunilor** anterior executate incluzând și cazul **instrucțiunilor triviale**.

UTILIZAREA MEDIEI ARMONICE ÎN EVALUAREA VITEZEI DE EXECUȚIE A PROGRAMELOR (HM)

- Se consideră că fiecare program execută O operaţii
- Se consideră că n programe execută cele nO operaţii în ΣT_i
- Viteza (rata) de execuţie a sistemului (P) este în acest caz dată de relaţia:

$$P = nO / \Sigma T_i = n / \Sigma(T_i/O) = n / \Sigma 1/P_i$$

unde $1/P_i$ este inversul vitezei de execuţie a programului i.

- Viteza de execuţie a unui sistem (măsurată în IPC – instrucţiuni per ciclu de $tact_{CPU}$) care rulează n benchmark-uri este egală cu media armonică a performanţelor individuale de procesare a fiecăruia dintre cele n benchmark-uri.

- **HM** elimină vârfurile (cazurile extreme care ar denatura rezultatul în cazul unei medii aritmetice) conducând la interpretări greşite a rezultatelor.

- Se recomandă calculul **HM** mai ales că fiecare benchmark în parte are un număr diferit de instrucţiuni şi evident un timp diferit de execuţie.

Probleme:

1. Se consideră două calculatoare desktop numite CSA având ISA de procesor MIPS şi care execută instrucţiuni la o frecvenţă de 2.5 GHz, respectiv CSB având ISA de procesor x86 şi care execută instrucţiuni la o frecvenţă de 3 GHz. În medie, programele executate pe arhitectură MIPS au o lungime mai mare, de 1.5 ori decât aceleaşi programe scrise şi executate pe arhitectură x86.
   Se cere:
   a) Ştiind suplimentar că pentru execuţia programului P1, o instrucţiune pe CSA durează 2 perioade de tact (CPI_{CSA}=2) şi instrucţiunile aceluiaşi program P1 executat pe CSB necesită 3 perioade de tact (CPI_{CSB}=3), care sistem de calcul execută mai rapid programul P1? Care este accelerarea (*speed-up*-ul) sistemului mai performant?
   b) Dacă însă, pentru execuţia programului P2, o instrucţiune pe CSA durează 1 perioadă de tact (CPI_{CSA}=1) şi instrucţiunile aceluiaşi program P2 executat pe CSB necesită 2 perioade de tact (CPI_{CSB}=2), care sistem de calcul execută mai rapid programul P2? Care este accelerarea (*speed-up*-ul) sistemului mai performant?

   Indicaţie:
   1a) Se consideră că programul P1 dacă se execută pe CSB cu arhitectură x86 (CISC) are n instrucţiuni. Rezultă că acelaşi program P1 impune execuţia pe pe CSA cu arhitectură MIPS (RISC) a $1.5n$ instrucţiuni. Calculăm timpul de execuţie aferent programului P1 pe cele două sisteme de calcul CSA şi CSB. Astfel:

   $T_{CSB}(P1)$ = n *instr* * *Ciclii de procesare per instrucţiune* * *Perioada de tact procesor*$_{CSB}$ =
   = n * 3 * $1/(3*10^9)$ [s] = n [ns - nanosecunde]

   $T_{CSA}(P1)$ = 1.5n *instr* **Ciclii de procesare per instrucţiune* * *Perioada de tact procesor*$_{CSA}$ =
   = 1.5n * 2 * $1/(2.5*10^9)$ [s] = $1.2n$ [ns - nanosecunde]

   Astfel, rata de procesare IR_{CSB} = n instr. / n [ns] = 1 instr. / 1[ns] şi
   rata de procesare IR_{CSA} = n instr. / 1.2n [ns] = 1 instr. / 1.2[ns] şi

rezultă Speed-up=IR_{CSB} / IR_{CSA} = 1.2x => programul P1 se execută de 1.2 ori mai rapid pe CSB decât pe CSA.

Verificaţi în aceeaşi manieră, dar în condiţiile de la punctul b) care dintre cele două sisteme de calcul este mai rapid.

2. Se consideră că dintre instrucţiunile unui program executat pe un procesor MIPS 10% sunt instrucţiuni de împărţire. Toate celelalte instrucţiuni (care nu sunt de împărţire) necesită 1 ciclu de tact pentru execuţie. Instrucţiunile de împărţire se execută în 50 de ciclii de tact (se mai numesc *mari consumatoare de timp*). Ce observaţi şi cum interpretaţi în doar două propoziţii după ce rezolvaţi cerinţele de la *a)* la *f)*?
   a) Determinaţi care este metrica CPI aferentă programului executat pe acest tip de procesor? (în câţi ciclii de tact se execută în medie un astfel de program)
   b) Ce procent de timp se pierde doar cu operaţiile / instrucţiunile de împărţire?
   c) Care este speed-up-ul obţinut dacă îmbunătăţim performanţa de procesare a instrucţiunilor de împărţire de 2 ori (numărul de ciclii de execuţie pentru o instrucţiune de împărţire se reduce la jumătate)?
   d) Care este speed-up-ul obţinut dacă îmbunătăţim performanţa de procesare a instrucţiunilor de împărţire de 5 ori (numărul de ciclii de execuţie pentru o instrucţiune de împărţire se reduce de 5 ori)?
   e) Aceeaşi întrebare ca la punctele anterioare c) şi d) dacă îmbunătăţim performanţa de procesare a instrucţiunilor de împărţire de 10 ori respectiv de 50 de ori?
   f) Cât va deveni speed-up-ul dacă instrucţiunile de împărţire vor deveni de un infinit de ori mai rapide (timp execuţie≈0 tacte de procesare)?

Bibliografie

[Sch02] Schmidt R., Notohardjono B. D., High-end server low-temperature cooling, IBM Journal of Research and Development, Volume 46 Issue 6, November 2002, Pages 739 - 751.

Competiții internaționale vizând probleme vechi și totodată actuale din Computer Architecture

- **Branch Prediction Is Not A Solved Problem:Measurements, Opportunities, and Future Directions (hard to predict – unbiased branches, rare branches, indirect jumps & calls)**
  - **5th JILP Workshop on Computer Architecture Competitions (JWAC-5): Championship Branch Prediction (CBP-5) - https://www.jilp.org/cbp2016/, in conjunction with: ISCA-43 http://isca2016.eecs.umich.edu**

- **Applying Deep Learning to the Cache Replacement Problem**
  - **2nd Cache Replacement Championship (CRC-2) - http://crc2.ece.tamu.edu, co-located with ISCA-44, Toronto, Canada, June 25, 2017**

- **Context-Base Computational Value Predictor with Value Compression**
  - **1st Championship Value Prediction (CVP-1) - https://www.microarch.org/cvp1/cvp1/program.html, in conjunction with: ISCA-45 http://iscaconf.org/isca2018/index.html, sponsored by Qualcomm**