Eight great Ideas in Computer Architecture

Cele 8 mari idei ale arhitecturii sistemelor de calcul au fost inventate in ultimii 60 de ani de proiectare a computerelor. Aceste idei sunt atat de puternice incat au rezistat mult dupa ce primul computer le-a folosit, iar arhitectii noi isi demonstreaza admiratia prin imitarea predecesorilor.

1. Design for moore’s Law

Singura constanta pentru arhitectii de sisteme de calcul este schimbarea rapida, care este indrumata in mare parte din Legea lui Moore. Conform acesteia resursele circuitelor integrate, se dubleaza la fiecare 18-24 de luni. Legea lui Moore a rezultat dintr-o predictie din anul 1965 a unei astfel de cresteri a capacitatii circuitelor integrate facute de Gordon Moore, unul dintre fondatorii Intel. Cum realizarea unei arhitecturi a unui system de calcul poate dura ani, resursele disponibile per chip se pot dubla sau chiar cvadrupla intre inceputul si terminarea proiectului. Arhitectii trebuie sa anticipeze ince punct se va afla tehnologia cand design ul va fi terminat, decat atunci cand acesta abia este inceput.

1. Use Abstraction to Simplify Design

Atat arhitectii de sisteme de calcul cat si programatorii au trebuit sa inventeze tehnici pentru a deveni mai productivi, pentru ca altfel timpul de design s-ar prelungi la fel de repede ca resursele despre care se vorbeste in Legea lui Moore. O tehnica majora a productivitatii pentru hardware si software este sa se foloseasca abstractii pentru a reprezenta design-ul pe nivele diferite; detaliile de nivel inferior sunt ascunse pentru a oferi un model mai simplu la nivele inalte.

1. Make the Common Case Fast

Facand cazul obisnuit(foarte intalnit) mai rapid va tinde sa imbunatateasca performanta mai bine decat optimizarea cazurilor rare. In mod ironic, cazul obisnuit este de cele mai multe ori mai simplu decat cazurile rare, si prin urmare este adesea mai usor de imbunatatit. Acest sfat de bun simt implica faptul ca stim care este cazul foarte intalnit, fapt posibil doar prin experimentare atenta si masuratori. (ex: este mai usor sa faci o masina mica mai rapid decat sa faci un minivan rapid)

1. Performance via Parallelism

Inca din vremurile timpurii ale folosirii sistemelor de calcul, arhitectii acestora au oferit modele(designs) care au mai multa performanta daca efectueaza operatii in parallel.

1. Performance via Pipelining

Un model particular de parallelism este atat de intalnit in arhitectura sistemelor de calcul incat merita propriul nume: pipelining. De exemplu, inaintea masinilor de pompieri, o “brigada cu galeti” ar fi raspuns la un incendiu. Orasenii formau un lant de oameni sa care o sursa de apa la foc, cand ar fi putut mult mai rapid sa mute galeti de apa de-a lungul lantului, in loc ca persoanele sa tot mearga si sa se intoarca.

1. Performance via Prediction

Urmand zicala “e mai bine sa iti ceri iertare decat sa iti ceri permisiunea” ultima mare idee este prezicerea. In unele cazuri, poate fi mai rapid in medie, sa ghicesti si sa incepi sa lucrezi decat sa astepti pana cand cunosti raspunsul, presupunand ca mecanismul prin care te recuperezi dupa o predictie gresita nu este prea scump, iar predictia este relativ precisa.

1. Hierarchy of Memories

Programatorii vor ca memoria sa fie rapida, mare, dar si ieftina, din moment ce viteza memoriei defineste performanta de cele mai multe ori, capacitatea limiteaza dimensiunea problemelor care pot fi rezolvate, iar costul memoriei, in zilele noastre, reprezinta un procent semnificativ din costul intregului computer. Arhitectii au descoperit ca pot sa se adreseze acestor cereri contradictorii prin ierarhia memoriei, cu cea mai rapida, mica si cea mai scumpa memorie per bit in topul ierarhiei, si cea mai lenta, mare si ieftina per bit la capatul acesteia. Memoria Cache da impresia programatorilor ca memoria principala este aproape la fel de rapida ca cea din topul ierarhiei si la fel de mare si ieftina ca cea din capatul ei.

1. Dependability via Redundancy

Computerele nu trebuie sa fie doar rapide, ci si de incredere. Din moment ce orice dispozitiv fizic poate esua, sistemele de incredere sunt realizate incluzand componente redundante care pot prelua in momentul in care un esec apare, dar si ajuta la detectarea acestora.