Računarstvo visokih performansi

VELJKO PETROVIĆ I GORANA GOJIĆ

3—Problem performansi

BRZO O BRZINI I OPTIMIZACIJI

Šta su performanse?

- Imamo dve moguće definicije:
  - Teoretske performanse.
  - Praktične performanse.
- Teoretske performanse su *apsolutni maksimum* koji neki hardverski sistem može da izvuče i meri se u broju *nekakvih* operacija u sekundi. Najčešće, jedinica je FLOPS—**FL**oating point **OP**eration per **S**econd.
- Računari kakve vi, realistično, imate imaju performanse koje se mere u desetinama gigaFLOPSa, ne računajući GPU.
- Najbrži računar? IBM Summit. 200 **petaflops.** Čudo šta 13MW može da uradi.

Šta su performanse?

- To je lepo, ali nama ne treba računar da troši struju i zvuči impresivno.
- Nama treba rešenje, i to dovodi do *praktičnih* performansi.
- Praktične performanse su, efektivno, koliko vremena treba da se dođe do rešenja.
- Mnogo su realističnije (pošto nas baš to zanima) ali dobiti ih je jako jako teško.
- Tipično se procenjuju na osnovu kalibracionog programa—Benchmark-a.

Kako programer zamišlja računar?

- Moj program ima nekakve podatke i sam kod.
- I jedno i drugo živi u memoriji.
- Kod se sastoji od atomskih operacija, *instrukcija* koje traju neku jedinicu vremena t_i.
- Procesor izvršava moje instrukcije, jednu po jednu.
- Ako hoću brži program, opcije su mi:
  - Manje instrukcija.
  - Kraće t_i.

Oh, sweet child of summer…

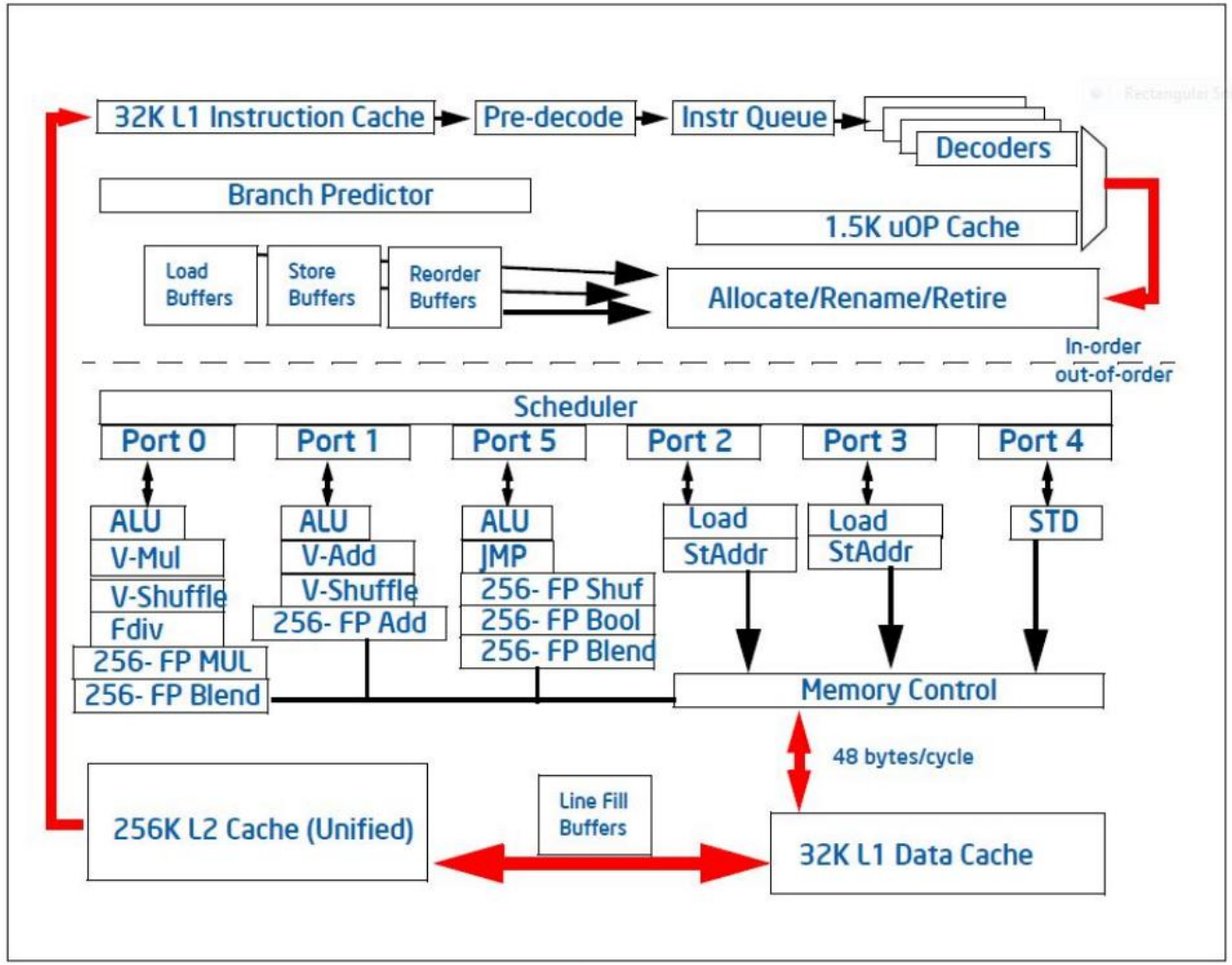
- …svi vi, znate nadam se da ovo nije tačno.
- Ali možda ne znate *koliko* nije tačno.
- Ipak, iako nije tačno ovo nije *potpuno* beskorisno.
- Minimizacija broja instrukcija je, generalno govoreći, dobar način da se program ubrza.
- Možete misliti o ovome kao o kontroli vremenske kompleksnosti algoritma.
  - Da li ste vi ovo radili?
- To je dobra ideja, ali ne svrha ovog kursa.

Dobro, šta ne valja sa ovom pričom?

- Vaš procesor ima u sebi, efektivno, više procesora.
- Ali svaki od *tih* procesora izvršava više stvari istovremeno.
- Takođe, ta istovremenost je kompleksna zahvaljujući pipelining-u.
- Takođe takođe, mehanizmi u računaru operišu na *kompletno* različitim vremenskim skalama.
- Takođe takođe…

Ovo je komplikovanije nego što izgleda.

- Računar se *jako* trudi da vam predstavi sliku da je samo instancirana Fon Nojmanova arhitektura i da je memorija lako i proizvoljno adresabilna.
- Lakše je tako programirati i većinu vremena *želite* tu iluziju, ali ne i kada hoćete da iscedite svaki poslednji dram performansi iz sistema.



Ovo je komplikovanije nego što izgleda.

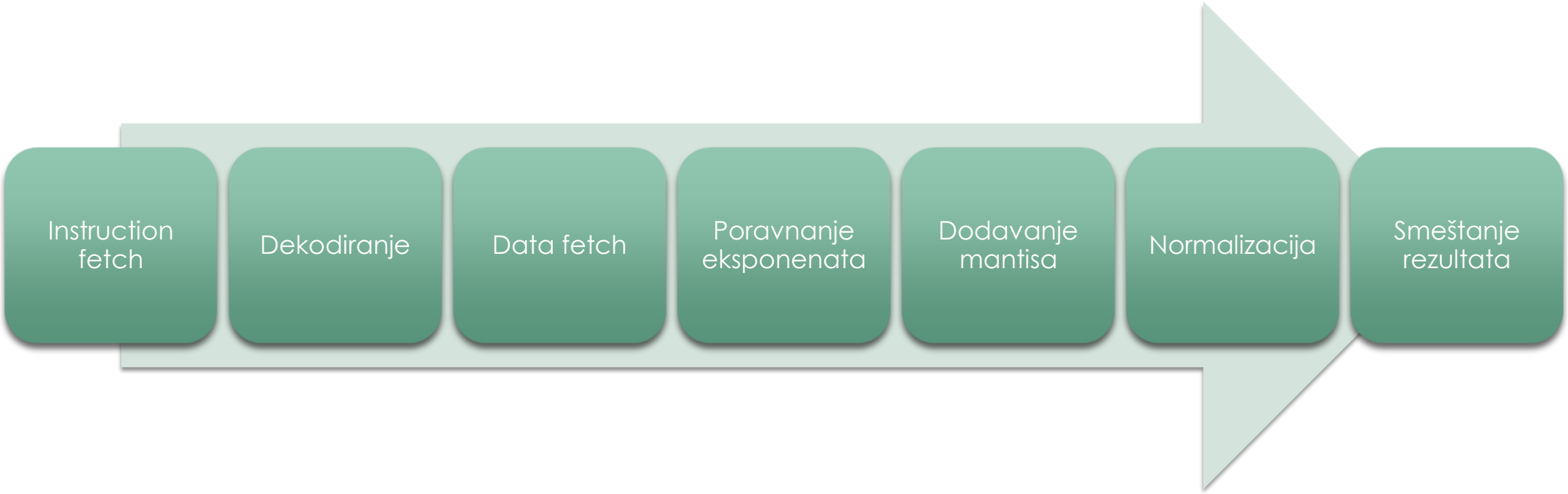
Šta su glavne komplikacije na *jednom* računaru?

- Ne zaboravite, ovde još pričamo o prostom računaru koji vam stoji na radnom stolu.
- Prvo, ima više jezgara.
- Drugo, instrukcije mogu da traju različiti broj ciklusa.
- Dalje ima paralelizam na nivou instrukcija (eng. Instruction Level Paralelism)
- Memorija ima striktnu hijerarhiju

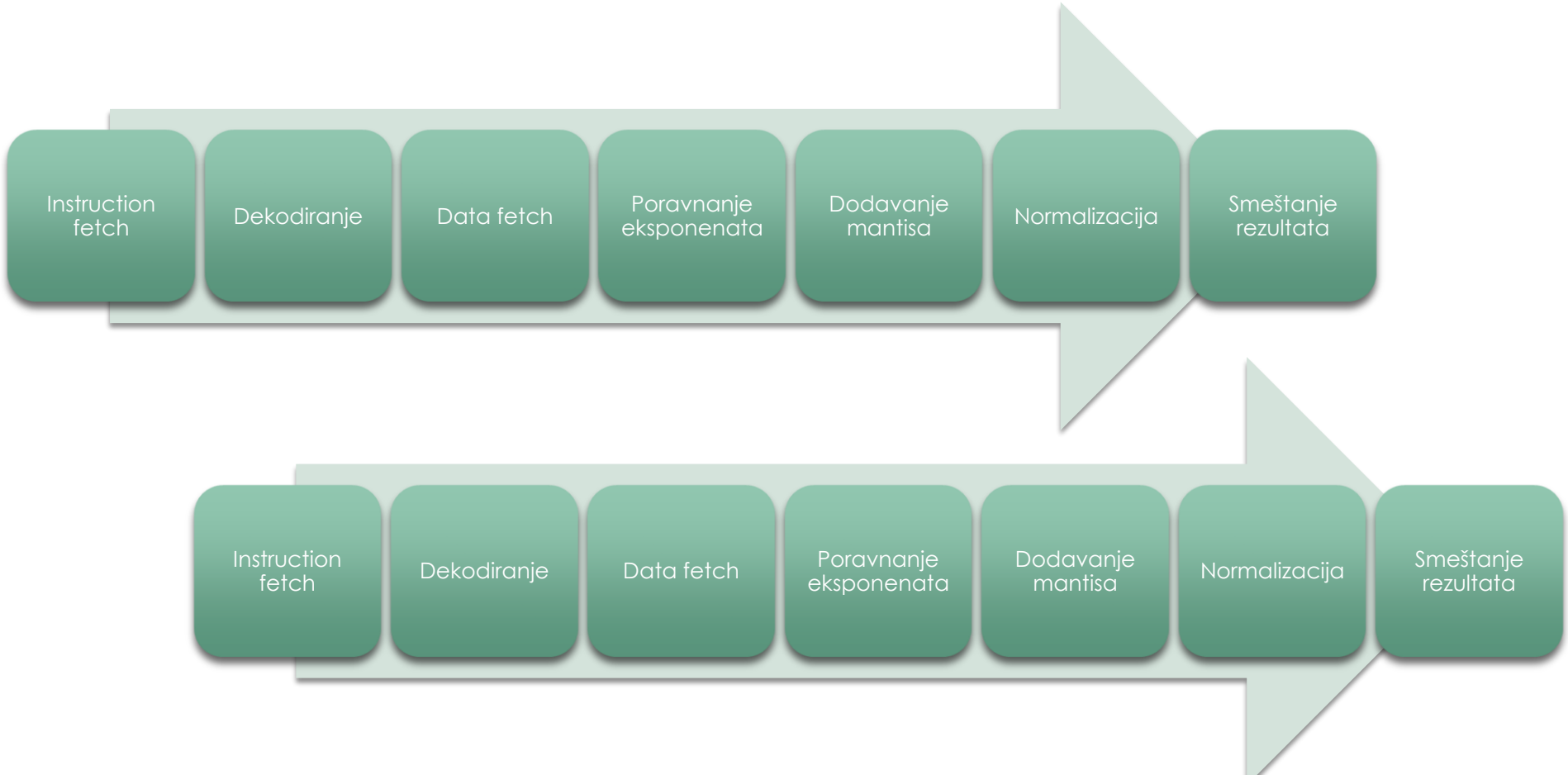
ILP

- Nezavisne instrukcije mogu da krenu da se izvršavaju u isto vreme, koristeći paralelne strukture u samom silikonu.
- Zahvaljujući sekvenci izvršavanja, više funkcija može da ide jedno za drugim u protočnom režimu (eng. "pipelining")
- Da ne bi bilo praznog hoda, procesor izvršava grane u vašem kodu *pre nego što se zna u koju će se ući.* Ako je pogodio kako treba, odlično, ako ne rezultat se baca i program se vraća u prethodno stanje.
- Da bi pipelining radio što bolje, instrukcije za koje je to moguće će biti izvršene u najoptimalnijem redosledu *ne vašem redosledu*.
- Podaci se dostavljaju iz nivoa memorije u nivo memorije spekulativno, tj. ako se *misli* da će možda trebati.

Pipeline


Instruction fetch
Dekodiranje
Data fetch
Poravnanje eksponenata
Dodavanje mantisa
Normalizacija
Smeštanje rezultata

Pipeline


Instruction fetch
Dekodiranje
Data fetch
Poravnanje eksponenata
Dodavanje mantisa
Normalizacija
Smeštanje rezultata
Instruction fetch
Dekodiranje
Data fetch
Poravnanje eksponenata
Dodavanje mantisa
Normalizacija
Smeštanje rezultata

Pipeline ubrzanje

- n — broj proruačna koje hoćemo
- l — broj koraka u procesnom toku
- τ — vreme za jedan ciklus sistemskog sata
- t(n) — vreme za n operacija
- s — vreme neophodno da se namesti da pipelining radi

Pipeline ubrzanje

Brzina serijskog izvršavanja

- $t(n) = nl\tau$

Brzina ILP izvršavanja

- $t(n) = [s + l + n\ - 1]\tau$

Hijerarhija memorije

Tip memorije	Red veličine	Brzina učitavanja
Registar	128 bajtova	Koliko i procesor
L1 Keš	~16 KB	Pola procesora
L2 Keš	~256 KB	Oko šestina procesora
L3 Keš	~8MB	Oko 12-ina procesora
Glavna memorija	~16GB	100 ciklusa sa oko 5% protoka
SSD disk	~512GB	Jako dugo.
HDD disk	~2TB	Večnost.

Keš

- *Nikad* nema dovoljno.
- Ko se seća Celerona, Durona, i sl.?
- U praksi, automatski mehanizmi pokušavaju da u kešu drže podatke koje nama trebaju.
- Ako, kada program zatraži podatak, on stoji u kešu *odlično*. Imamo ubrzanje.
- Ako ne, imamo omašaj, ond. "cache miss."

Katalog omašaja

- Neizbežan
  - Kada prvi put tražimo podatke.
- Kapacitetski
  - Kada nema više mesta.
- Konflikt
  - Kada mapiramo (keš menja memoriju, tj. lokacije u kešu su ubrzane memorijske lokacije sa tačke gledišta adresiranja), mapirali smo dve stvari na isto mesto.
- Invalidacija
  - Više jezgara se posvađalo oko toga šta je najsvežija verzija nekog podatka.

Keš blok

- Instrukcije ne mogu da direktno adresiraju keš
- I dalje misle da pričaju sa glavnom memorijom
- Ovo je česta apstrakcija odgovorna i za, npr. memory-mapped I/O
- Iza kulisa, mikrokontroler procesora uzima podatke iz memorije i smešta ih u keš u jedinicama fiksne veličine (blokovima).
- Tipično, 128 bajtova. To znači da dobijamo ceo taj komad memorije hteli mi to ili ne.
- Zatim se beleži koji deo memorije je mapiran na koji deo keša i, kada ponestane prostora, menja se najdavnije korišćeni deo. LRU
  - Ovo je laž. Više o tome kasnije.

Koja je praktična primena znanja o keš blokovima?

- Pakovanje podataka.
- Ako prolazimo kroz niz element po element, kada učitamo prvi element, uz njega dolazi N sledećih džabe.
- Ako procesiramo svaki element, onda to je to.
- Ali šta ako je ovo niz tačaka u 3D prostoru a nas samo zanima X vrednost.
- Imamo jako puno bačenih učitavanja.
- Ako znamo kako će neki podaci biti procesirani, isplati se da se upakuju tako da podaci koji se zajedno koriste budu blizu.

Array stride

- Recimo da hoćemo da saberemo dva niza kompleksnih brojeva.
- To znači (ako koristimo double preciznost) da nam treba 16 bajtova po broju.
- Keš linija je, recimo, 128.
- To znači da bi trebalo da je brže da se brojevi sa sabiranje prepletu u jedan niz.
  - Re(A0)
  - Im(A0)
  - Re(B0)
  - Im(B0)
  - Re(A1)
  - …

Address alignment

- Lukaviji možda mogu da primete da ja mogu da adresiram *bilo koju adresu* u glavnoj memoriji na bajt nivou, čak i ako radim na nivou reči.
- Da li to znači da nekako, magično, ima poravnanje između keša i memorije?
- Ne. Multi-bajt vrednost može vrlo lako da bude u dva keš bloka.
- Ovo je *spektakularno* loše po performanse.
- Ponekad, kompajler je dovoljno pametan da to otkloni.
- A ako nije?

```c
#include <stdio.h>
#include <stdlib.h>

int main(){
        double* a;
        a = malloc(1024*8 + 8);//Niz 1024 double vrednosti.
        printf("%p\n",a);
        if((size_t)a % 8 != 0) { //detektovano neporavnanje
                a = (double*)(( ((size_t)a >> 3) << 3));
        }
        printf("%p\n", a);
        free(a);
}
```

Do try this at home…

Rezultat eksperimenta

- Na mom GCC7, ovo je beskorisno. Već dobijam poravnanu memoriju.
- *Jako* puno zavisi od kompajlera, i često morate kod da tetošite predprocesorskim direktivama da dobijete ono što želite.

Malo sam lagao o kešu…

- Rekao sam ranije da se beleži region memorije i lokacija u kešu
- To… baš i nije tačno.
- To bi zahtevalo tkzv. asocijativan keš. Ovi su spori.
- Ono što se koristi u praksi je k-struki asocijativan keš.
- To znači da postoji transformaciona funkcija koja mapira lokaciju u memorji na lokaciju u kešu na više mogućih načina. Tipično od 2 do 8.
- Onda, u slučaju konflikta u mapiranju, koristi se jedna od dodatnih lokacija oslobođena po LRU principu.

Šta je sve ovo trebalo da me nauči?

- Osim malo o arhitekturi računara ima još i ovo:
- Kako se nešto implementira interaguje jako komplikovano sa hardverom procesora i računara uopšte da proizvede performanse.
- Stoga, programiranje performantnog koda može biti jako izazovno.
- A sve ovo je na samo *jednom* računaru…

4—HPC Uvod

ŠTA KADA PROCESOR JEDNOSTAVNO NIJE DOVOLJNO BRZ

Paralelizam

- Prethodna sekcija je pokazala da su performanse teške, ali je sva bila opsednuta time da se iz *jednog* procesora izvuče maksimum.
- Budući da se naš kod, na kraju dana, izvršava na nekom procesoru, negde, to nije loša ideja i uvek će biti relevantno, ali šta kada 100% nekog procesora nije dovoljno brzo?
- Postoje praktične granice gigahercaži koju možemo da dobijemo iz čipa
  - Bakar
  - Brzina svetlosti

Paralelizam

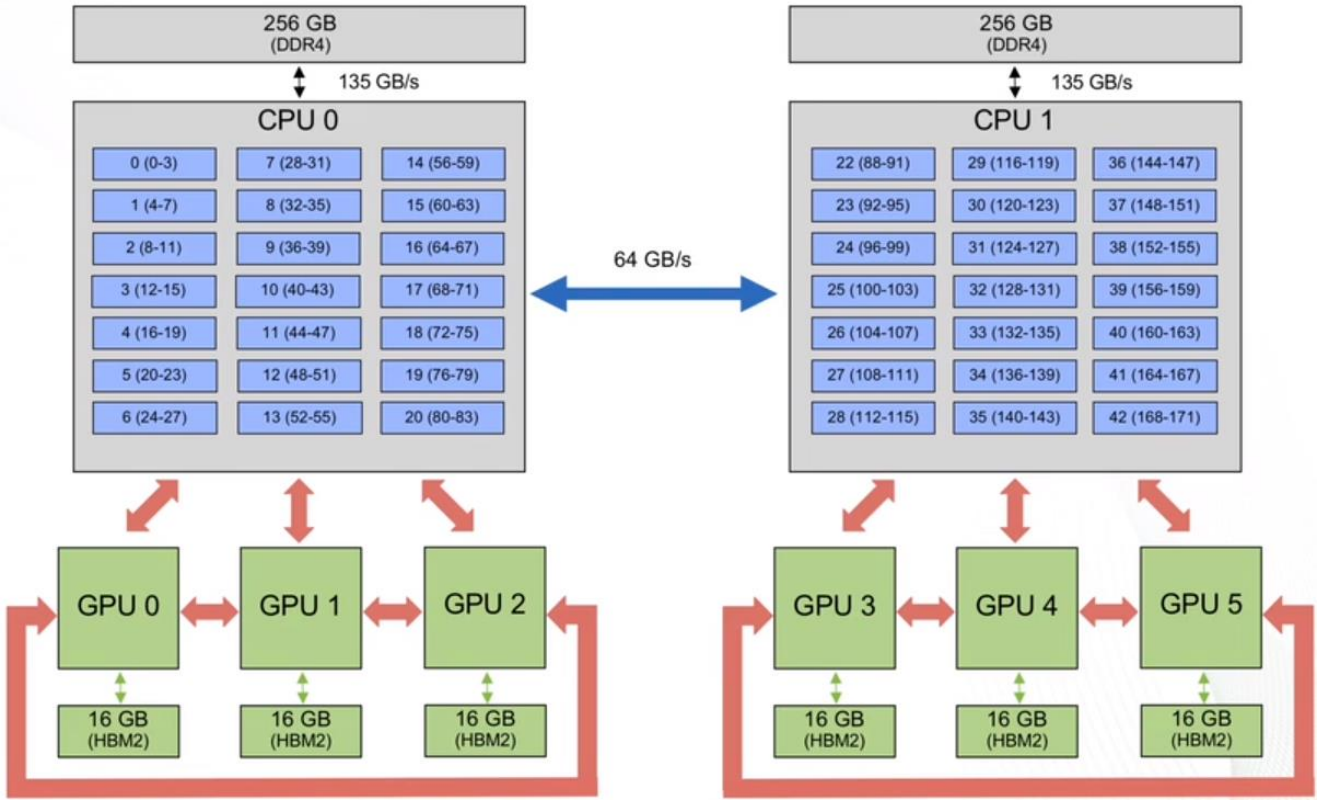
- Zbog ovoga superračunari danas nisu (i verovatno nikad više neće biti) jedan jako moćan procesor.
- Ono što čini superračunar super jeste broj procesorskih elemenata.
- Da bi se broj procesorskih elemenata iskoristio kako treba, potrebno je pisati kod koji je *paralelan,* tj. izvršava više stvari istovremeno.
- Ovo nije paralelizam na nivou instrukcije, koliko je paralelizam na nivou algoritma.
- Priroda algoritma dramatično utiče na to koliko je lako odn. teško izvršiti paralelizaciju.

Šta je naš posao?

- Da dobijemo odgovor jako jako brzo.
- Da, ali kako?
- Napadamo problem sa dve strane:
  - Arhitektura
  - Algoritam
- Dakle, treba da napravimo mašinu koja ima jako efektan dijapazon procesora koji brzo komuniciraju i imaju šta im treba da ostvare blizu svom teoretskom maksimumu.
- Sa druge strane treba da upravljamo tom mašinom i podelimo algoritme na takav način da se adekvatni delovi algoritma izvršavaju na pravom mestu radi brzine.

Anatomija jednog superračunara

- Superračunar tipično ima neki broj čvorova.
- Čvor se može zamisliti kao jedan računar.
- U našem klasteru to stvarno *jesu* računari.
- Naš klaster ima 4 uređaja.
- IBM Summit, ima malo više. Specifično 4604 više.
- Svaki čvor je jako moćan i sadrži više procesora (2) sa više jezgara (21) gde svako jezgro podržava 4 nezavisna izvršavanja i više izuzetno moćnih GPU-ova (6).
- Takođe ima oko 1600GB memorije po čvoru.



Jedno jezgro

Mnogo paralelizama

- Ovde ima jako puno stvari koje podržavaju paralelizam
- Izazov jeste napraviti kod koji ima odgovarajući posao za svaki paralelizam koji hardver nudi
- Neke stvari su taman za rad na jednom jezgru ili jednom procesoru
- A neki poslovi se najbolje dele između individualnih čvorova gde je komunikacija između njih izuzetno retka.
- Različite tehnologije su dobre za različite nivoe paralelizma. Gledano iz ptičije perspektive

Mnogo paralelizama

Harverski nivo paralelizma	Tehnologija
Unutar jednog procesora	OpenMP/pThreads
Između čvorova	OpenMPI
Na GPU uređajima	OpenACC

Skaliranje

- Skaliranje je kako ukupne ostvarne performanse zavise od veličine sistema.
- Tj. ako povećamo računar dva puta koliko dodatnih performansi dobijemo od toga?
- Idealno dva, da, ali…
- Skaliranje ima dva tipa
  - Slabo
    - Uniformni rast veličine sistema, memorije i problema.
  - Jako
    - Veličina problema ostaje ista, skalira se veličina sistema za povećanje brzine.

Šta smeta skaliranju?

- Koristi se mnemonik SLOW za faktore koji sprečavaju da sistem dostigne svoj teoretski maksimum. SLOW su:
- Starvation
  - Nema dovoljno posla da se uposle svi resursi sistema.
  - Možda sistem može da radi 600 svari istovremeno, ali ako trenutno postoji samo 6 nezavisnih zadataka, sistem radi na 1% svojih performansi.
- Latency
  - Sistem može da bude veliki, i ako informacija sa jednog kraja sistema bude neophodna na drugom, čekanje na nju proizvodi značajno usporenje. Setite se dijagram od ranije i različitih protoka podataka.
- Overhead
  - Sav taj kod koji deli podatke i vodi računa ko radi kada šta i integriše rezultate itd. itd. itd. oduzima neko vreme i neke resurse da se izvrši.
- Waiting
  - Čim ima više niti izvršavanja može doći do problema nadmetanja ("contention") oko deljenih resursa. Ovo se rešava čekanjem. In a stunning turn of events, waiting turns out to be bad for performance. Who knew?

5—Kratka istorija

KAKO SMO STIGLI OVDE?

Mehaničko računanje

- Želja za mašinom koja računa umesto nas, ili barem proces čini lakšim je verovatno samo par minuta mlađa od samog koncepta računanja.
- Rano računanje je, u stvari, bilo samo po sebi fundamentalno vezano za nekakvo pomoćno ustrojstvo.
- Drevni Rimljani su imali brojeve koji su bili prilično teški za mehaničku manipulaciju
- Mislim, koliko je LXVII puta XI? A da ne prebacite prvo u arapske brojeve?
- Rimljani su imali metod koji je uključivao pažljivo napravljenu tablicu i kamenčiće.
- Deminutivska množina reči za 'kamen' na latinskom je 'calculi'
- Odatle kalkulator, kalkulisanje, itd. itd.

Mehaničko računanje

- Ovo je nastalo, toliko da su ljudi koji su radili sa novcem (te mnogo računali) u kasnosrednjevekovnoj Italiji uvek imali pri ruci klupu sa ucrtanom šemom za račun.
- Termin za klupa je bio 'banca'
- Kasnije su prešli na novi, divni metod za računanje koji ne zahteva klupu no upotrebljava čudne strane cifre. Taj metod su zvali po iskvarenom imenu kreatora 'algorisam'
- Mušterije nisu verovale ovakom algorismičnom bankarstvu i hteli su klupe nazad. Naročito omrznuta je bio potpuno novi simbol—nula. Toliko je bila omrznuta da je arapsko ime za nju—al sifr—ušlo u skoro sve evropske jezike.
- Kao 'šifra.'

Mehaničko računanje

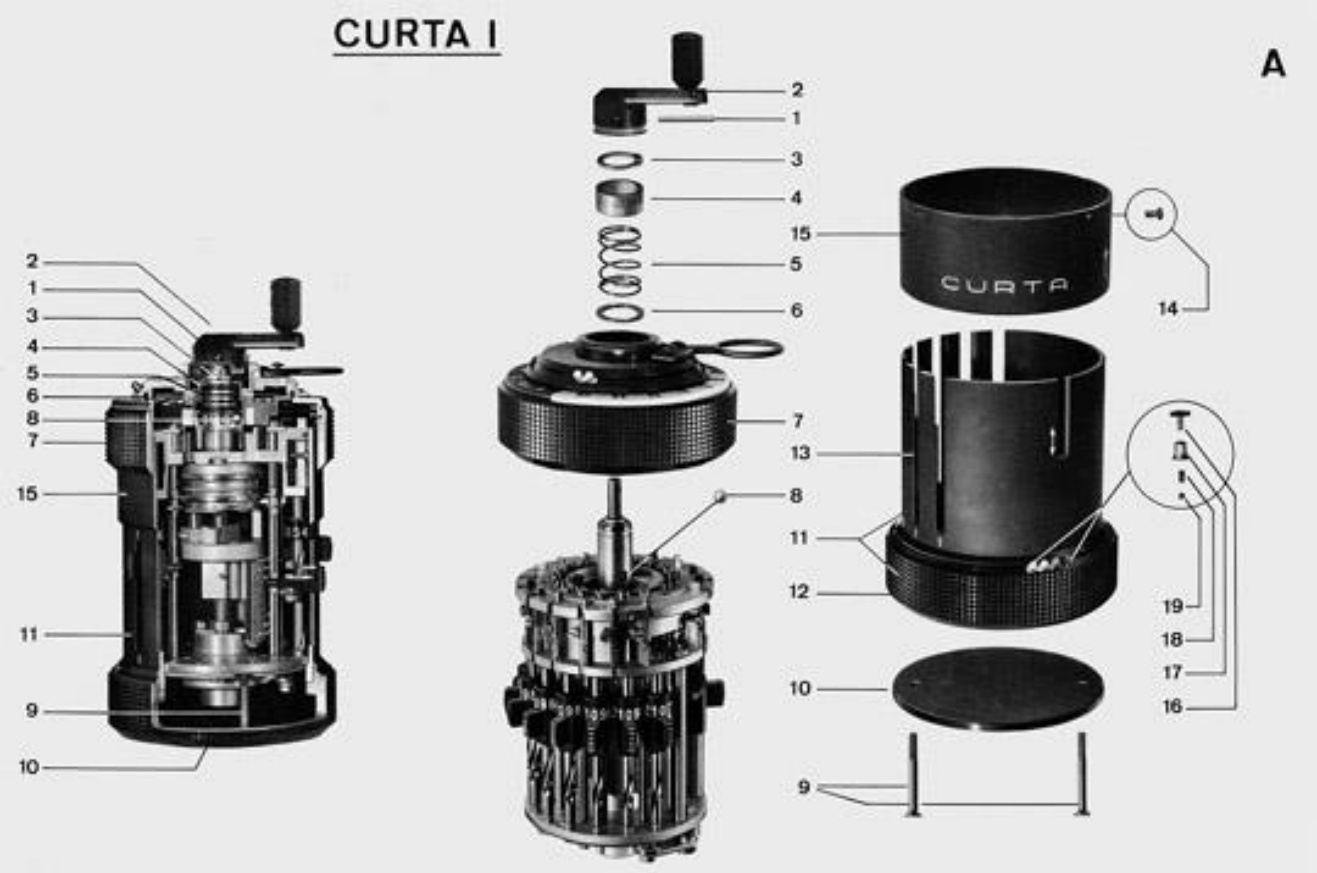
- Ne možemo, očigledno da se oslobodimo mehaničkih računala.
- Prva mehanička računala su bila fundamentalno samo računaljke.

Abakus

Logaritmar


2
3
2
3
3
1
Centaine de mille
dizaines de mille
mille
Centaine
dizaine
Nombre Simple

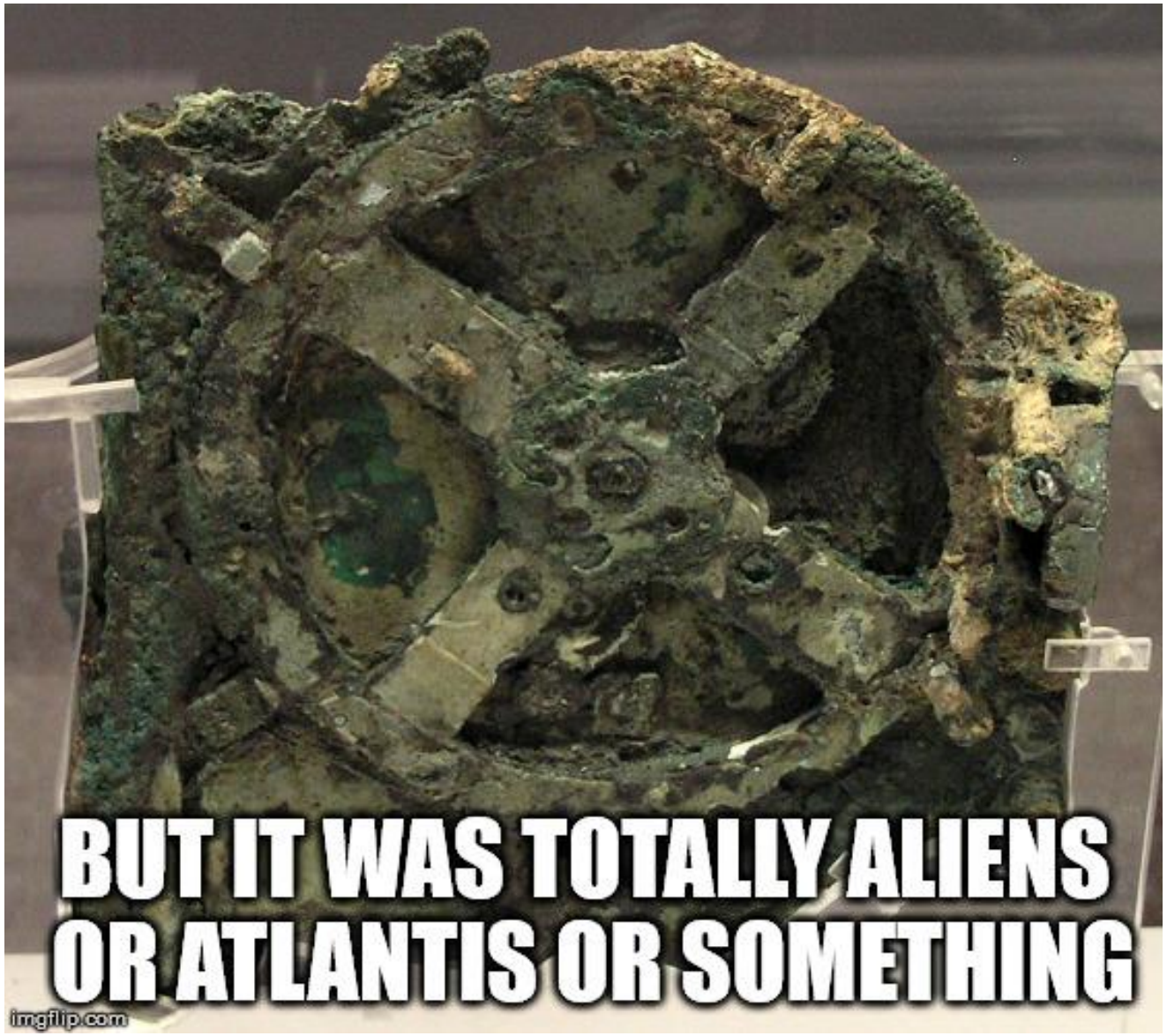
Paskalina


CURTA I
A
2
1
3
4
5
6
8
7
15
11
9
10
2
1
3
4
5
6
7
8
11
12
10
9
15
13
14
19
18
17
16
CURTA

Curta

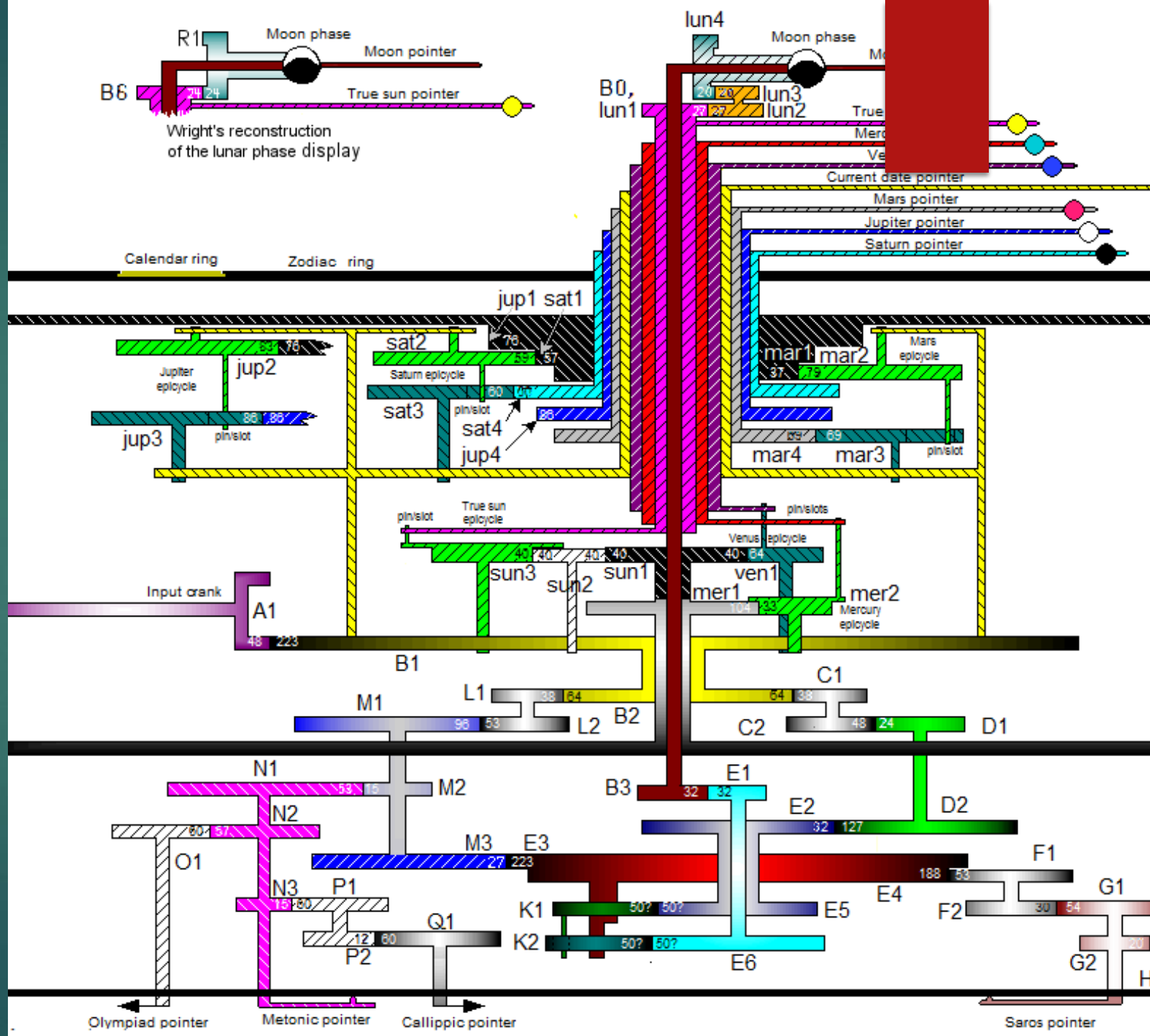
Računari?

- Ne. Nikako.
- U računaru ovo odgovara više ALU nego celom računaru.
- Šta je onda prvi računar?



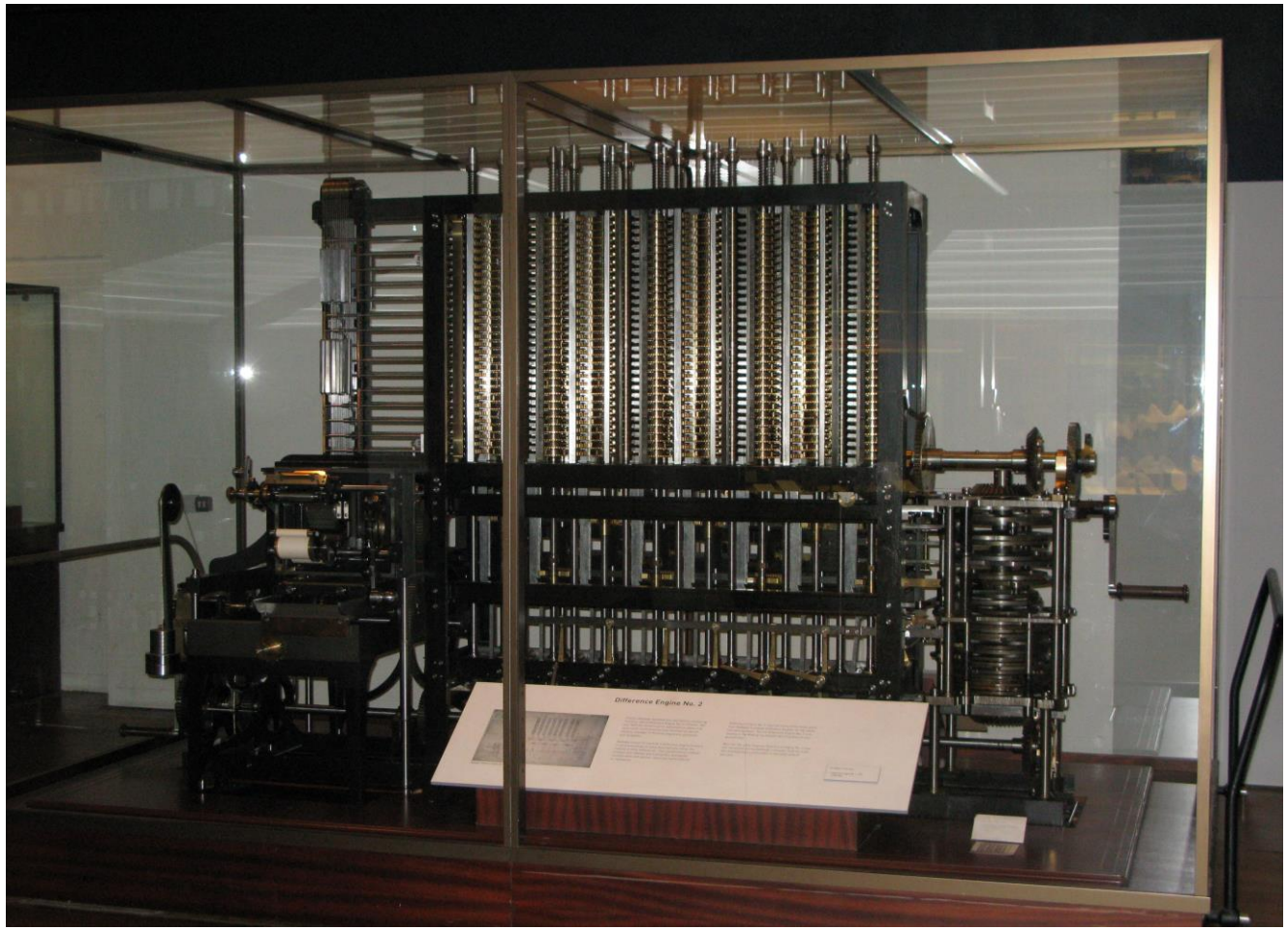
I'm not saying it was aliens…

Antikiteranski mehanizam



Analogni računar

- Mislite o tome kao o kompleksnoj fizičkoj inkarnaciji matematičke funkcije
- Možemo da menjamo parametre, ali priroda funkcije je ista
- Jeste računar ali nije *programabilan*.

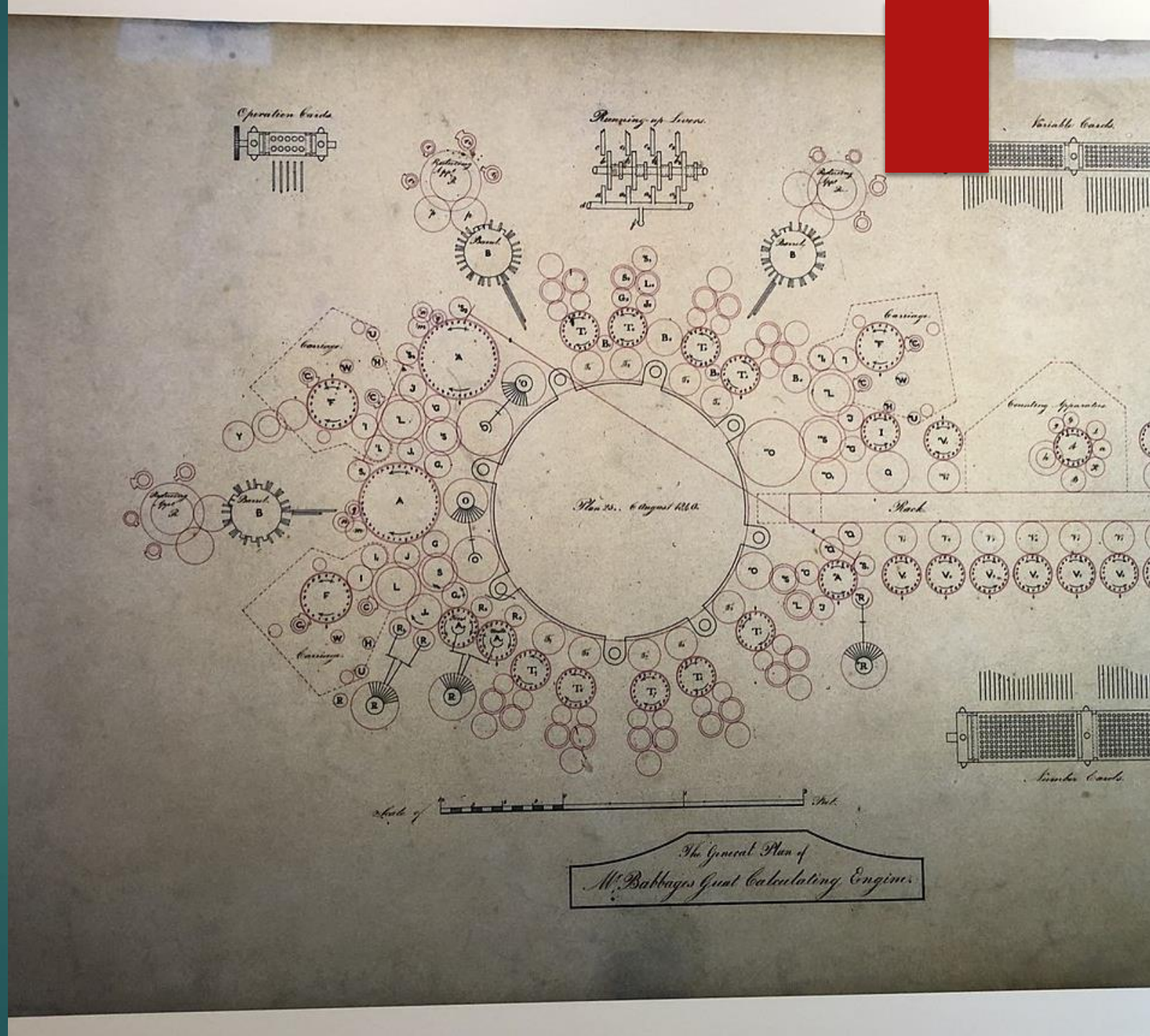


The Difference Engine

Računar?

- Ne još.
- Ovo je fizička inkarnacija *gomile* funkcija.
- Šta ima da je programabilno?

The Analytic Engine



Prvi pravi računar

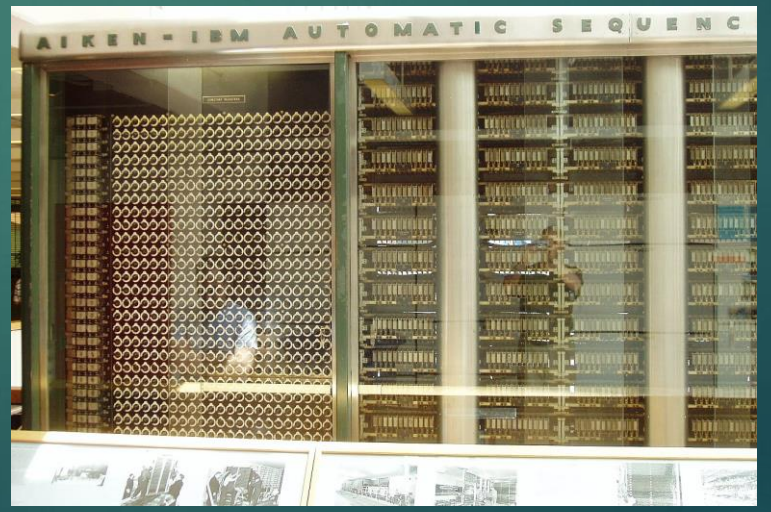
- …ali samo na papiru.
- Memorija, programi, opšta namena.
- Nikada nije napravljen, ali moderne studije ukazuju da je mogao biti uz *ogromnu* investiciju.

Da se uozbiljimo

- Šta je sa istorijom elektronskih super-računara?
- Tradicionalno, deli se u 'epohe' koje karakteriše dominacija određenih tehnologija.

Epoha I—elektromehanička era

- Zuse Z1 (1938)
- Harvard Mk1(1944)

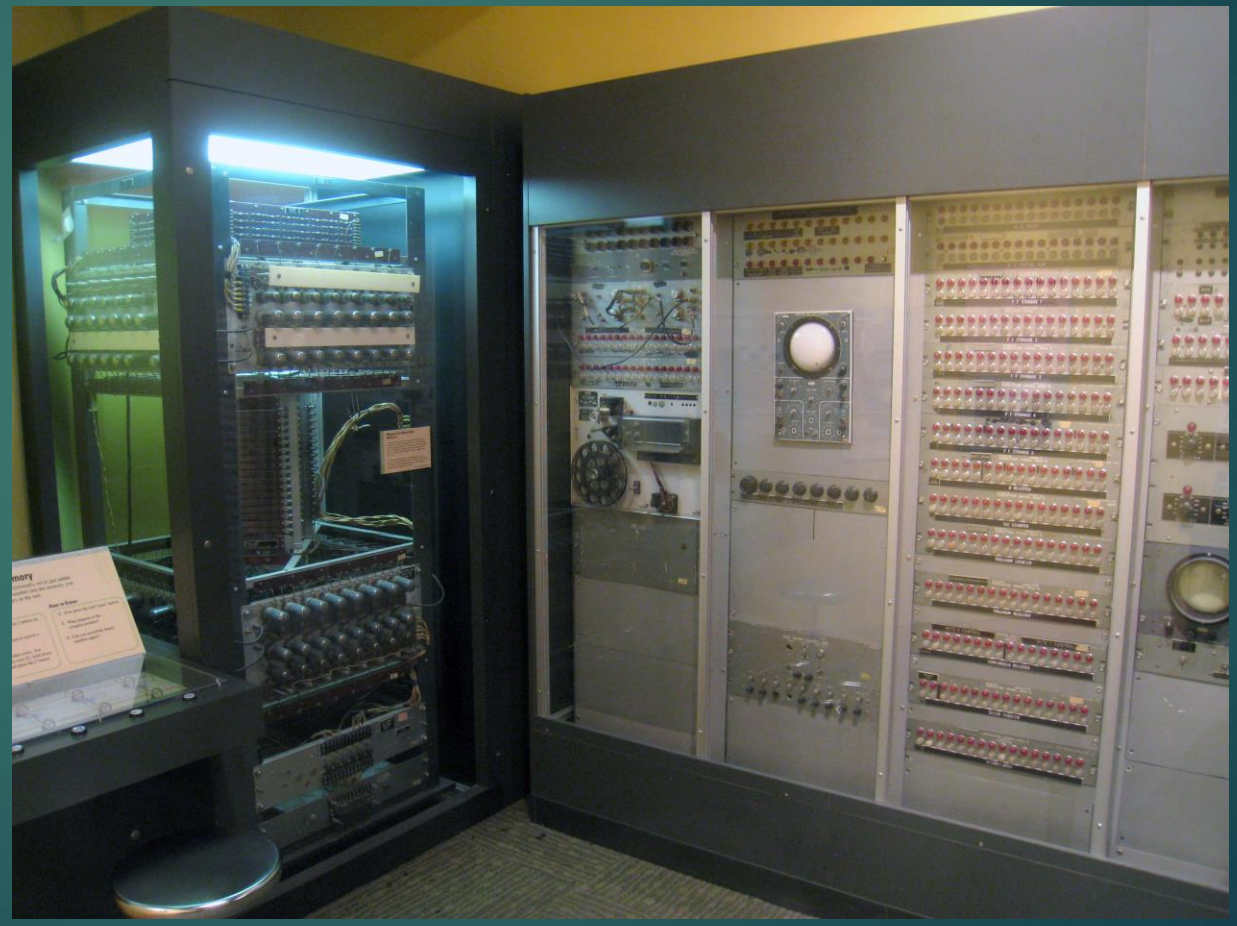


Epoha I—elektromehanička era

- Brzine od čak jedne instrukcije u sekundi!
- Bušene kartice za I/O
- Još nema programskih jezika kao takvih
- Čerč i Tjuring postavljaju teoretske osnove računara

Epoha II—Fon Nojmanova Arhitektura i Vakumske cevi

- ENIAC
- EDSAC (1949)
- Colossus
- IBM 704
- UNIVAC (1951)



Epoha II—Fon Nojmanova Arhitektura i Vakumske cevi

- Vrhunske mašine ere postižu do 10 KIPS
- 4KB memorije
- U ranom dobu koriste se živina kola za memoriju
- Kasnije magnetna jezgra
  - Ovo je ostavilo traga do danas—ako vam je ikada pukao program u Linux-u i ostavio 'core dump,' sada znate odakle ime.
- U ovom periodu:
  - Fon Nojman postavlja osnove moderne arhitekture računara.
  - Klod Šanon postavlja osnove informatike.

Epoha III—Paralelizam na nivou instrukcije i uspon tranzistora

- Era počinje sa TX-0 računarom i vodi preko DEC PDP-1 i IBM 7090 do vrhunca treće epohe
- CDC 6600 (1965)

Epoha III—Paralelizam na nivou instrukcije i uspon tranzistora

- CDC6600 je imao
  - 1 MFLOPS!
  - 10 MHz takt!
  - 10 logičkih jedinica!
  - Prvi ILP!
  - Jedan od prvih uređaja koji se zvao „superkompjuter“

Epoha IV—Vektorski procesori i integrisana kola

- Računar koji je obeležio ovu epohu je legendarni Krej-1 (1976)
- Karakteriše ga *izuzetno* dugačak pipeline.

Epoha IV—Vektorski procesori i integrisana kola

- Krej-1 je mogao
  - 80 MHz!
  - 160 MFLOPS!
  - 8.39MB memorije!
  - 303MB diska!

Epoha V—SIMD i spor uspon mikroprocesora

- SIMD je jedna od fundamentalnih HPC arhitektura po Flinovoj taksonomiji (vidi kasnije)
- Podelimo podatke na blokove, a onda radimo istu stvar svakom bloku podataka.
- SIMD — Single Instruction Multiple Data
- Problem sa SIMD-om jeste što su algoritmi bili *fantastični* za neke stvari i potpuno beskorisni za sve ostalo.
- SIMD (kasnije, kasnije) i dalje živi—postoji način na koji je svaki GPU u stvari široka SIMD implementacija.
- NEC SX-2
  - Prvi računar da probije GFLOPS barijeru

Epoha VI—Mnogo Procesora

- Touchstone Paragon (1994)
- IBM SP-2
- Thinking Machines Corporation CM-5 (1992)
- Prvi moderni superkompjuteri
- Prosleđivanje poruka i odvojena memorija, po prvi put.
- Takođe, prvi put se pojavio potrošački klaster (commodity cluster)
  - UC Berkley NOW
  - Beowulf
    - PC + Linux + Ethernet + MPI = Supercomputing For Everyone

Epoha VII?

- Mi smo ovde.
- I dalje su dominantne tehnologije iz šeste epohe ali uz dodatak ekstremne heterogenosti.
- U jednom čvoru imamo i SIMD i shared-memory i message-passing.
- Moglo bi se reći da ovo predstavlja odrastanje HPCa.

Dalje?

- Masivne online mreže.
- 3D čipovi i sintetički dijamant
- Neuroprocesori i domain specific arhitekture
- Kvantni računari