Multiprocesorski sistemi

Uvod u paralelno računarstvo

Marko Mišić, Andrija Bošnjaković, Milo Tomašević

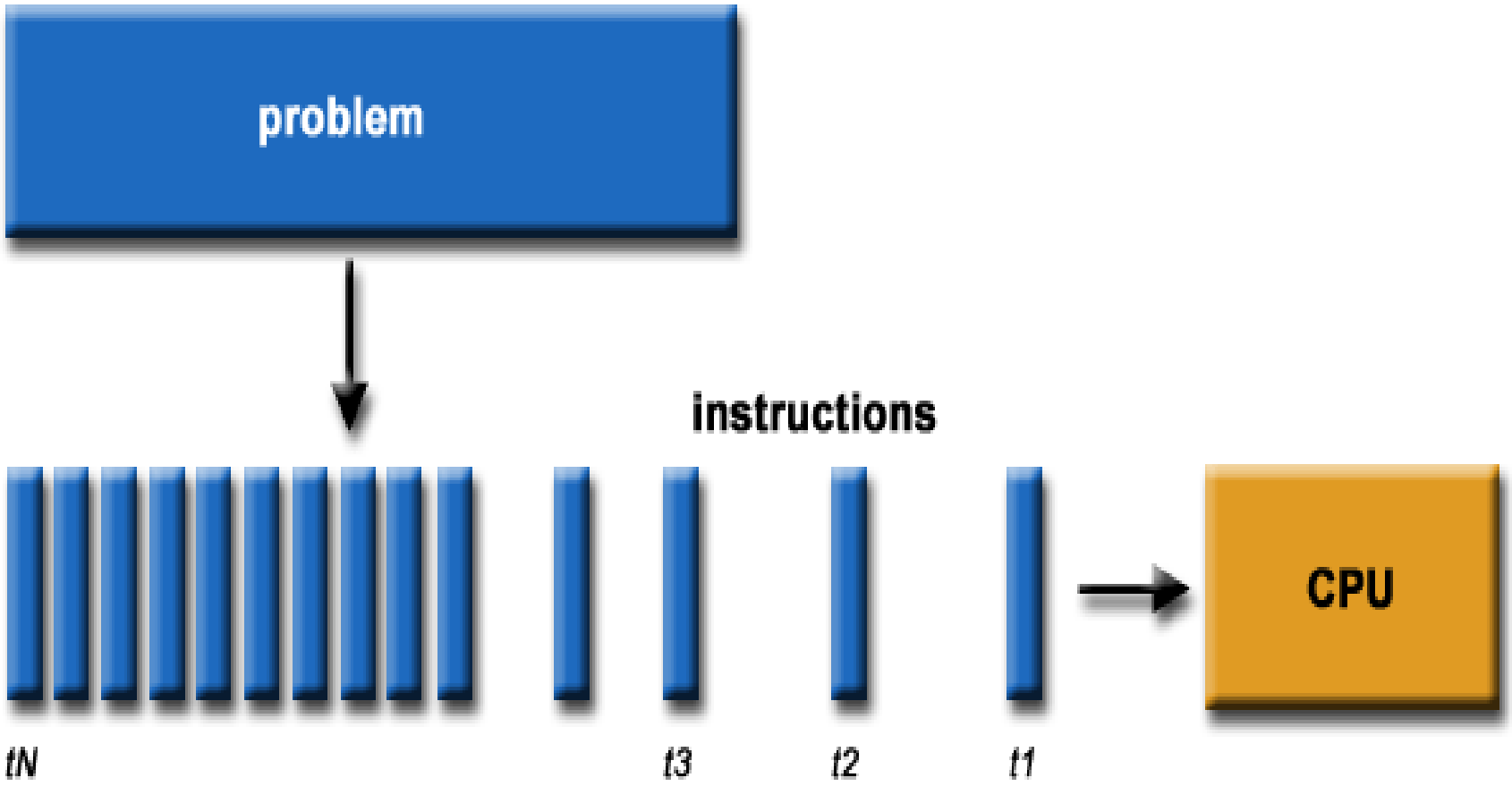
13S114MUPS, 13E114MUPS, 13M114MUPS

2024/2025.

Šta je paralelno računarstvo? (1)

- Ranije je softver pisan za sekvencijalno izvršavanje
  - Izvršava se na pojedinačnom računaru, sa jednim procesorom
  - Problem se razdvaja na više diskretnih serija instrukcija
  - Instrukcije se izvršavaju redom, jedna za drugom
  - Samo jedna instrukcija se može izvršavati u jednom trenutku

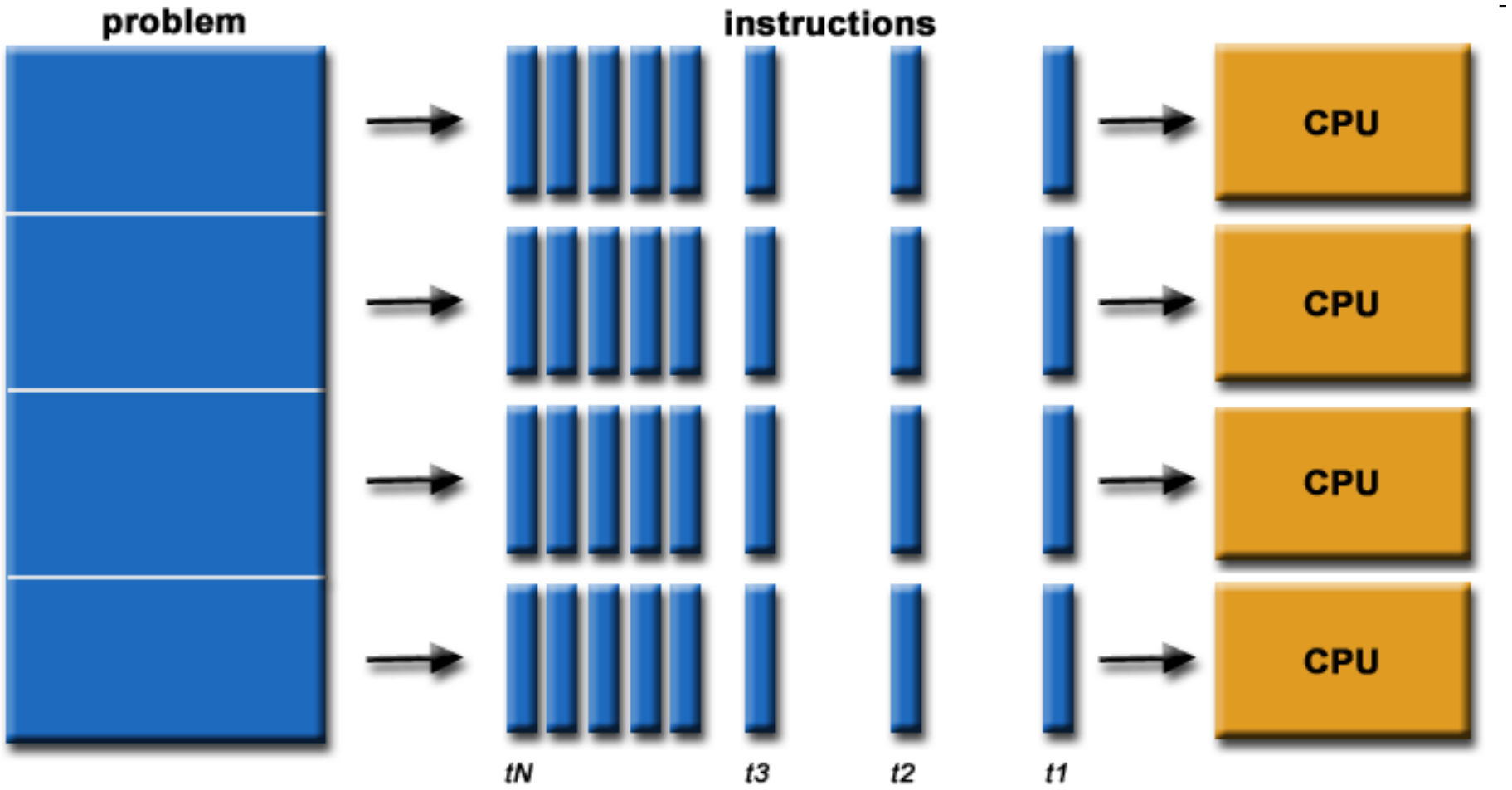
Šta je paralelno računarstvo? (2)



Šta je paralelno računarstvo? (3)

- Paralelno računarstvo predstavlja istovremeno korišćenje više računarskih resursa u cilju rešavanja nekog problema
  - Softver se izvršava na više procesora
  - Problem se razdvaja na više delova koji se mogu rešavati uporedno (u paraleli)
  - Svaki deo problema se dalje razdvaja na serije diskretnih instrukcija
  - Instrukcije iz svakog od delova se izvršavaju istovremeno na različitim procesorima

Šta je paralelno računarstvo? (4)



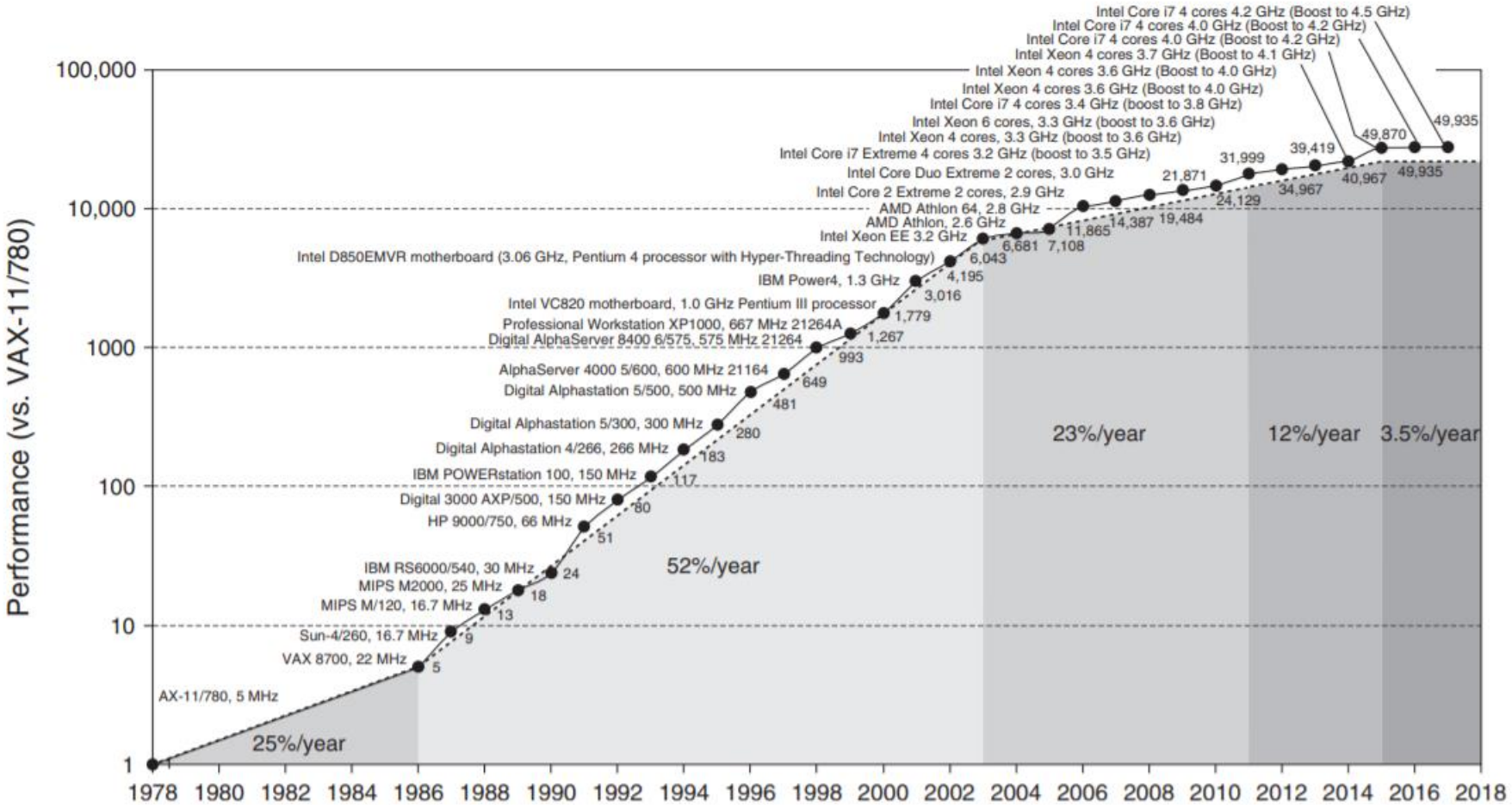
Šta je paralelno računarstvo? (5)

- Računarski resurs može biti
  - Pojedinačni računar sa više procesora
  - Određen broj računara povezanih mrežom
  - Kombinacija oba navedena
- Poželjne osobine problema koji se rešava
  - Mogućnost da se podeli na delove koji se mogu istovremeno rešavati
  - Mogućnost da bude rešen u kraćem vremenu korišćenjem višestrukih resursa

Računarski sistemi (1)

- Ogroman napredak za 70-tak godina
  - Iste performanse 500$ (danas) = 50M$(1993)
  - Tehnologija (predvidljivo)
  - Arhitektura (manje predvidljivo)
- Posledice:
  - Ogromno povećanje procesne snage i mogućnosti
  - Poboljšanje *Cost/Performance*
    dovelo do novih klasa računara (PC, WSC, mob)
  - Dominacija sistema zasnovanih na mikroprocesorima
  - Razvoj HW platformi utiče na razvoj SW
    (kompromis između produktivnosti i perfomanse)
  - Sve intenzivnije multimedijalne aplikacije
- Međutim, postoje određeni trendovi usporavanja…

Računarski sistemi - rast performansi



Računarski sistemi (2)

- *Moore-ov* zakon više ne važi
  - 1965, broj tranzistora se duplira svake godine
  - 1975, ... za svake dve godine
  - Danas, ... na svakih 20 godina
- Kombinacija uzroka:
  - Usporavanje *Moore-ovog* zakona i *Dennard-ovog* skaliranja
    - Povećanje gustine tranzistora na čipu i konstantna potrošnja energije
  - Nepromenjena snaga mikroprocesora
  - Zamena snažnih jednojezgarnih procesora sa višejezgarnim
  - Ograničenja *Amdahl-ovog* zakona
- Danas, dupliranje performansi na svakih 20 godina
- Pravac poboljašanja odnosa performanse cena
  - Domenski-specifične arhitekture

Paralelno procesiranje

- Promena principa razvojne filozofije favorizuje paralelno procesiranje
  - Energija je jeftina, tranzistori su skupi =>
    tranzistori su jeftini, energija je skupa (*power wall*)
  - FLOP je spora, L/S op je brza =>
    FLOP je brza, L/S je spora (*memory wall*)
  - Prevodioci i arhitektura omogućavaju sve veći ILP =>
    dobici od ILP sve manji (*ILP wall*)
  - Monolitni uniprocesori interno pouzdani, greške na pinovima =>
    ispod 65 nm drugačije
  - Skaliranje uspešnog HW bloka =>
    efekti dužine linija, preslušavanja, šuma, pouzdanosti, ...

Paralelno procesiranje

- Promena principa razvojne filozofije favorizuje paralelno procesiranje
  - Performanse uniprocesora x2 svakih 1.5g => usporenje trenda (možda čak na 5g)
  - Primarni dobitak od povećanja takta => primarni dobitak od povećanja paralelizma
  - *Power wall* + *memory wall* + *ILP wall* !

Paralelno procesiranje

- Prošlost ... sadašnjost ... budućnost
- Veliki napredak – tehnologija i koncepti
- Paralelni računar je *skup* procesnih elemenata koji sarađuju da bi *brzo* rešili *veliki* zadatak (A&G,89)
- Omogućava rešavanje velikih problema intenzivnim računskim i memorijskim zahtevima
- Namena:
  - Veća propusna moć (*throughput*) - više poslova istovremeno
  - Ubrzanje - paralelizacija aplikacije
  - Poboljšanje odnosa "*cost-effectiveness*" – manje ulaganja za bolje performanse
  - Veća pouzdanost

Paralelno procesiranje

- Bitne karakteristike:
  - Alokacija resursa:
    - koliko procesora?
    - kakva je snaga procesora?
    - kako su povezani?
    - koliko memorije i kakva je hijerarhija?
  - Pristup podacima, komunikacija i sinhronizacija
    - kako elementi sarađuju i komuniciraju?
    - gde se podaci nalaze i kako se prenose između procesora?
    - koje su apstrakcije i primitive za kooperaciju?
  - Performanse i skalabilinost
    - kakve su posledice na performanse?
    - kako sve to skalira?

Paralelno procesiranje

- Paralelizam:
  - Kvalitativna alternativa za poboljšanje performansi
  - Ima ga na svim nivoima projektovanja sistema (čak i u jednoprocesorskim sistemima)
  - Sve značajniji u obradi informacija
  - Zavisnost od aplikacije i tehnologije

- Glavne prepreke u SW:
  - Ogromne dosadašnje investicije u sekvencijalne programe
  - Nenaviknutost na paralelno programiranje
  - Dva nivoa programera
    - Produktivnost (90%)
    - Efikasnost (10%)

Paralelno procesiranje

- Istorija –
  različite i inovativne arhitekture i organizacije,
  često vezane za nove programske modele
- Brzo sazrevanje uz ograničenja tehnologije
  - Današnji i sutrašnji mikroprocesori su multiprocesori - CMPs
  - Laptop i super-računar koriste iste CPU!
  - Tehnološki trendovi utiču na konvergenciju različitih pristupa
- Trendovi tehnologije čine paralelno procesiranje nezaobilaznim
- Razumevanje osnovnih principa
  i projektnih kompromisa
  - Komunikacija, replikacija, sinhronizacija, balansiranje …

Klase računara

- Personalni mobilni uređaji (PMD)
  - Pametni telefoni, tablet računari
  - WWW, multimedijalne aplikacije
  - Cena, energetska efikasnost, realno vreme
- Desktop i laptop računari
  - Ogromno tržište (više od pola - laptop)
  - Obično se ovde javljaju najnoviji mikroprocesori
  - Širok opseg aplikacija
  - Odnos cena/performanse (CPU/GPU)
- Serveri
  - Računarski servis većeg obima i pouzdanosti
  - Dostupnost, skalabilnost
  - Propusni opseg (*tpm*)!

Klase računara

- Klasteri/ WSC (*Warehouse Scale Computers*)
  - klasteri – skupovi više servera vezanih na LAN
  - vrlo veliki klasteri (desetine hiljada servera) - WSC
  - koriste se za interaktivni "*Software as a Service* (SaaS)" - pretraživanja, društvene mreže, igre, kupovina ...
  - dostupnost , potrošnja, odnos cena/performanse
  - superračunari imaji isti red cene,
    ali računski zahtevne neinteraktivne velike aplikacije,
    pa bitne FLOP performanse, brze interne mreže, itd.
- IoT/ugrađeni (*embedded*) računari
  - koriste se u uređajima široke upotrebe (često 'pametnim')
  - širok spektar perfomansi i cena, nekad ograničen HW i SW
  - vrlo bitna cena
    (postići zahtevane performanse za datu cenu)

Klase računara

Feature	Personal mobile device (PMD)	Desktop	Server	Clusters/warehouse-scale computer	Internet of things/ embedded
Price of system	$100–$1000	$300–$2500	$5000–$10,000,000	$100,000–$200,000,000	$10–$100,000
Price of microprocessor	$10–$100	$50–$500	$200–$2000	$50–$250	$0.01–$100
Critical system design issues	Cost, energy, media performance, responsiveness	Price-performance, energy, graphics performance	Throughput, availability, scalability, energy	Price-performance, throughput, energy proportionality	Price, energy, application-specific performance

- Prodaja 2015.
  - 1.6 mlrd. PMD
  - 275 miliona PC
  - 15 miliona servera
  - 19 mlrd. EP

Paralelizam

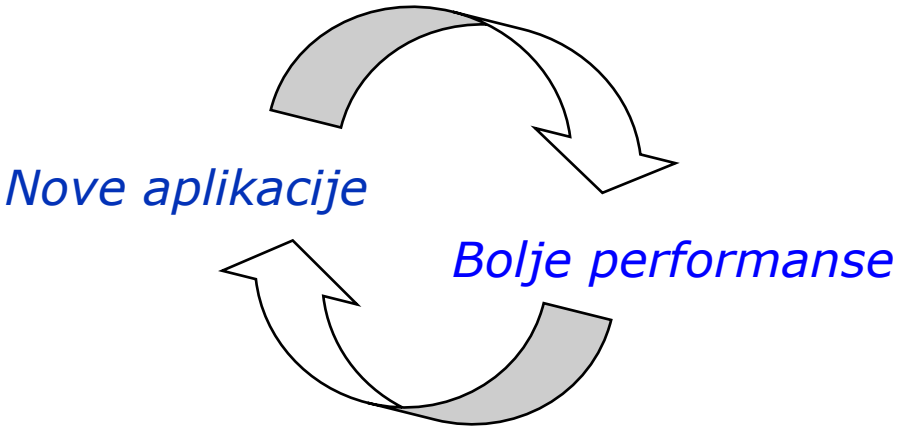
- Klase paralelizma u aplikacijama
  - Paralelizam na nivou podataka (DLP)
  - Paralelizam na nivou zadataka (TLP)
- Računari ove dve klase paralelizma koriste kao:
  - Paralelizam na nivou instrukcije (ILP)
    - Uz pomoć prevodioca i HW resursa
  - Vektorske arhitekture/GPUs
    - Koriste DLP na nivou iste instrukcije
  - Paralelizam na nivou niti (TLP)
    - DLP ili TLP u interativnim nitima
  - Paralelizam na nivou zahteva (RLP)
    - Nezavisni zadaci specificirani od programera ili OS

Paralelno procesiranje

- Zahtevi aplikacije – neprestane i rastuće potrebe za procesnom snagom računara
- Trendovi tehnologije
- Trendovi arhitekture
- Ekonomičnost, performanse, skalabilnost, ...

Trendovi aplikacije

- Zahtevi aplikacija za performansama podstiču unapređenja u HW, što omogućava nove aplikacije, ...
  - Posledica - eksponencijalno povećanje performansi mikroprocesora
  - Jak uticaj na paralelne arhitekture
    - najzahtevnije aplikacije



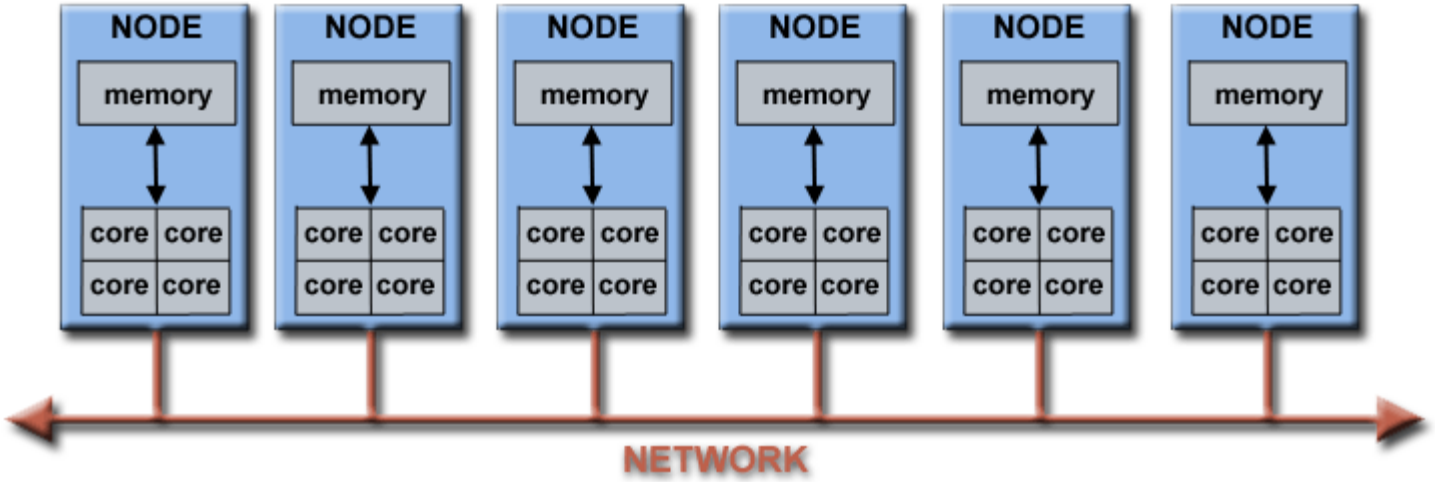
- Opseg zahteva za performansama
  - "piramida" platformi sa progresivnim porastom cene i perfomansi

Moderni paralelni računari (1)

- Skoro svi moderni *stand-alone* računara podržavaju neku formu paralelizma na nivou hardvera:
  - Imaju više funkcionalnih jedinica
    - Više nivoa keševa u paraleli (L1, L2)
    - Veći broj jedinica za obradu skokova, dekodovanje instrukcija, rad sa celim brojevima i brojevima u pokretnom zarezu
    - Grafički procesori i drugi koprocesori
  - Imaju više izvršnih jedinica – jezgara
  - Mogu da izvršavaju više hardverski podržanih niti istovremeno

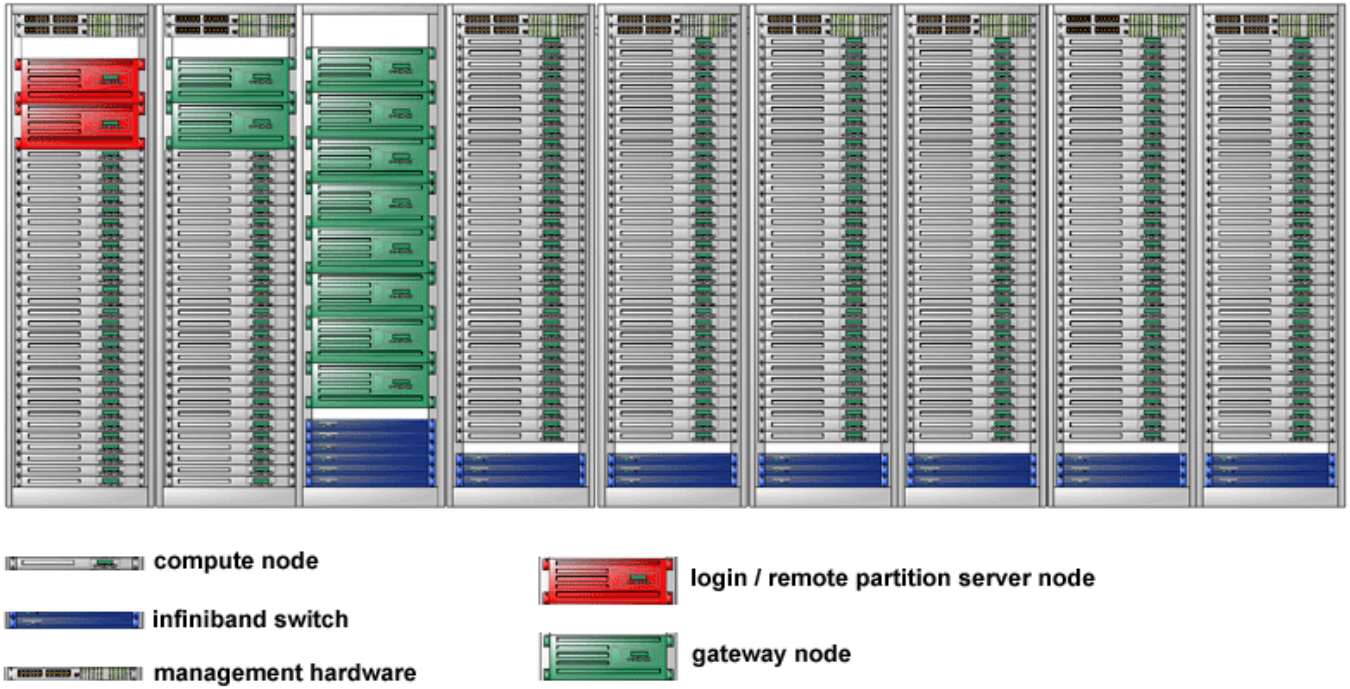
Moderni paralelni računari (2)

- Interkonekcione mreže se koriste da povežu pojedinačne računare u paralelne računarske klastere



Moderni paralelni računari (3)

- Arhitektura tipičnog klastera
  - Svaki računski čvor je multiprocesorski sistem za sebe
  - Više računskih čvorova je povezano Infiniband mrežom
  - Čvorovi specijalne namene su takođe multiprocesori
    - login, storage i gateway čvorovi



Zašto koristiti paralelno računarstvo? (1)

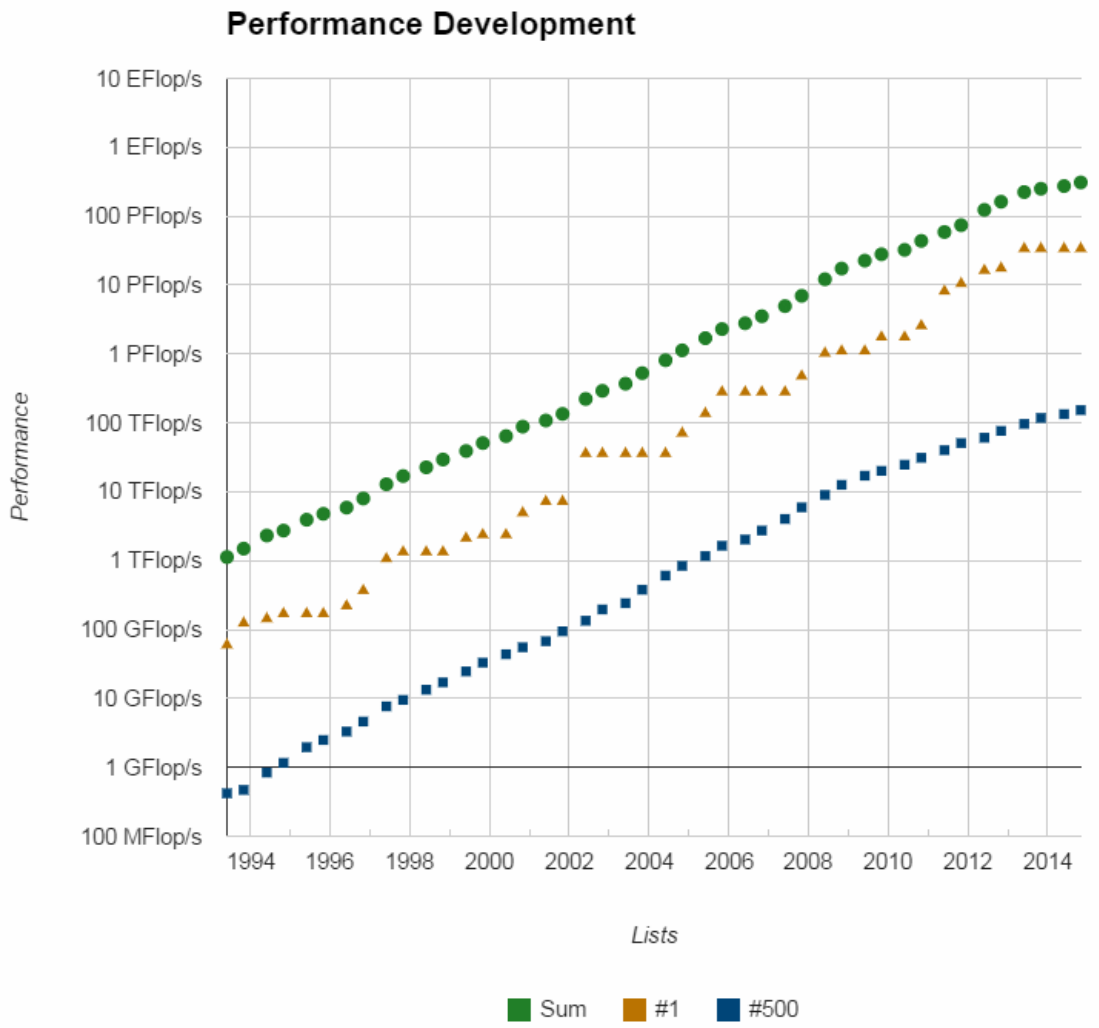
- Glavni razlozi
  - Ušteda na vremenu i troškovima (jeftine komponente)
  - Mogućnost rešavanja većih i složenijih problema
  - Mogućnost rešavanja više problema istovremeno
    - Omogućavanje konkurentnosti
  - Korišćenje sve dostupnijeg paralelnog hardvera
- Drugi razlozi
  - Mogućnost korišćenja nelokalnih slobodnih resursa (putem mreže ili celokupnog Interneta)
  - Prevazilaženje memorijskih ograničenja

Zašto koristiti paralelno računarstvo? (2)

- Ograničenja jednoprocesorskih sistema
  - Brzina prenosa signala na čipu
    - ograničena brzina prenosa bakarnih žica
  - Ograničenja minijaturizacije
    - Trenutno 32/22nm postupak, očekuje se zid na 9-11nm
  - Ekonomska neisplativost daljeg razvoja
- Trendovi poslednjih godina ukazuju da je paralelizam budućnost računarstva
  - Brže mreže, napredak na polju distribuiranih sistema, multiprocesorski desktop računari, GPU računarstvo...

Zašto koristiti paralelno računarstvo? (4)

- Top500 lista
- Do 2020. godine, Exascale computing



Primeri korišćenja paralelnog računarstva

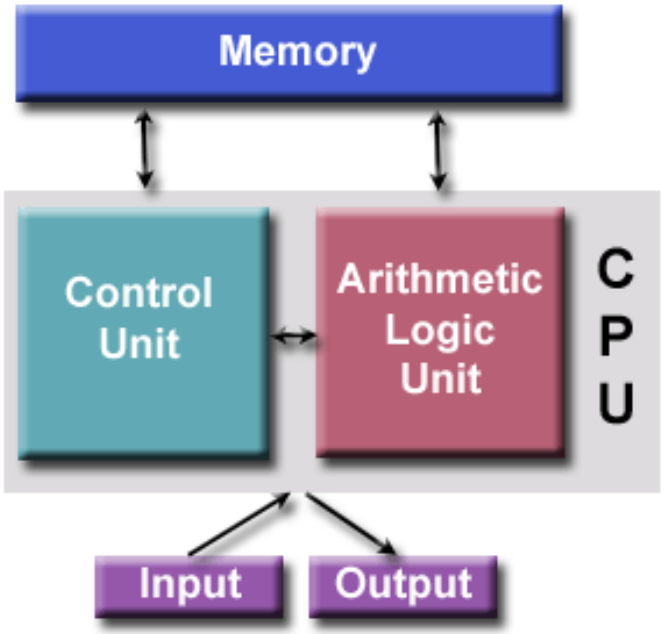
- Primena u nauci i inženjerstvu
  - "Grand challenge" problemi
  - Simulacije u fizici, hemiji, geologiji, biologiji i tehnici
    - Hemijske i nuklearne reakcije, ljudski genom, seizmičke aktivnosti, prognoza vremena
- Komercijalne aplikacije
  - Web pretraživači i web orijentisani servisi
    - Big data, data mining, machine learning, deep learning
  - Istraživanje nafte i gasa
  - Medicinska dijagnostika i farmaceutika
  - Virtuelna realnost
  - Mrežne multimedijalne tehnologije

Osnovni koncepti i podela paralelnih računara

Von Neumann arhitektura (1)

- Računarski model koji se koristi preko 40 godina
- Procesor izvršava program koji se sastoji od sekvence operacija čitanja i pisanja po memoriji
- Osnove dizajna
  - Memorija se koristi da se čuvaju instrukcije i podaci
  - Programske instrukcije su kodirani podaci koji govore računaru da uradi nešto
  - Podaci su informacije koje program koristi pri izvršavanju
  - Procesor dohvata instrukcije i podatke iz memorije, tumači instrukcije i onda ih sekvencijalno izvršava

Von Neumann arhitektura (2)



- Paralelni računari i danas slede ovaj bazični dizajn
  - Na nivou pojedinčanih, funkcionalnih jedinica

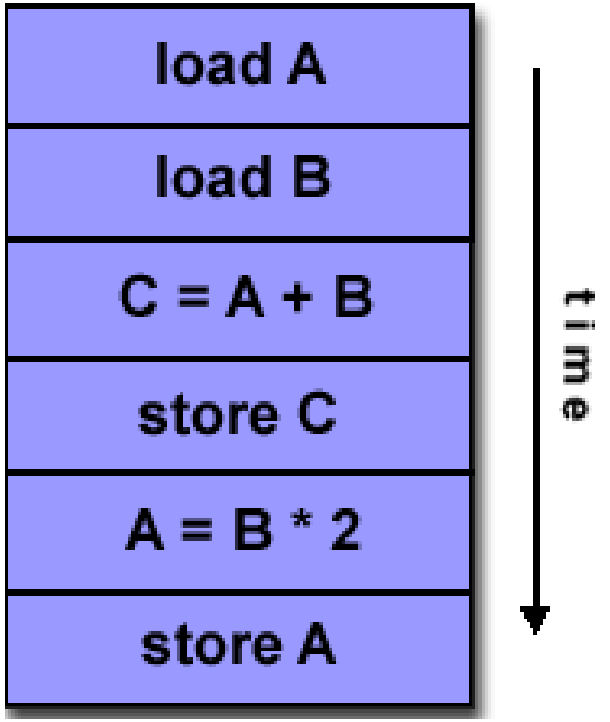
Flynn klasična taksonomija (1)

- Jedna od najkorišćenijih klasifikacija paralelnih računara
- Multiprocesorske arhitekture se dele prema dve nezavisne dimenzije: instrukcija (Instructions) i podataka (Data)
  - Svaka od ove dve dimenzije može imati dva stanja: Single i Multiple
  - Na osnovu ovoga su izvedene četiri moguće klasifikacije paralelnih računara

Flynn klasična taksonomija (2)

SISD Single Instruction, Single Data	SIMD Single Instruction, Multiple Data
MISD Multiple Instruction, Single Data	MIMD Multiple Instruction, Multiple Data

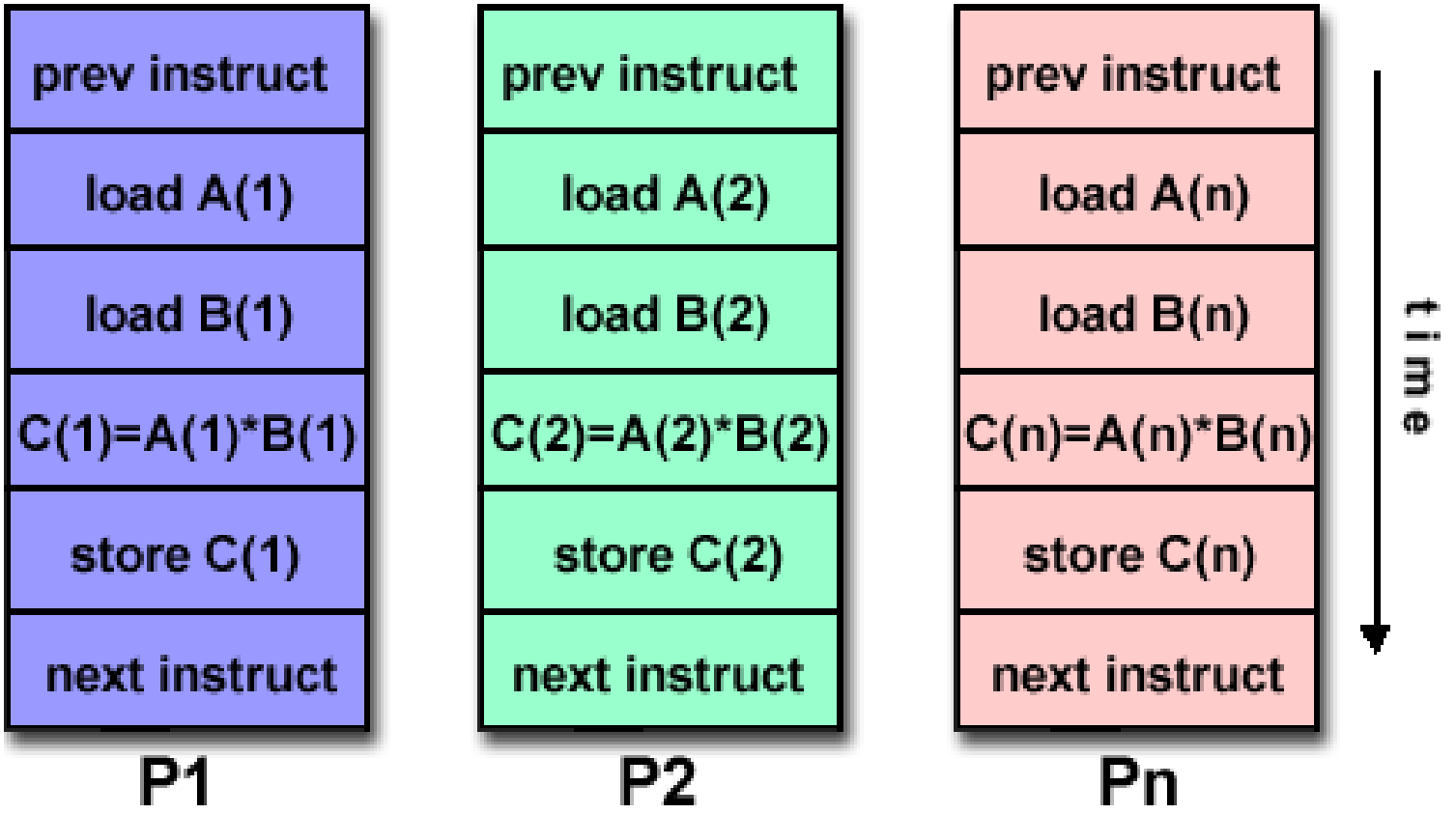
Single Instruction, Single Data (SISD) (1)



Single Instruction, Single Data (SISD) (2)

- Sekvencijalni računar
- Samo jedna instrukcija se izvršava u jednom ciklusu procesora (Single Instruction)
- Samo jedan tok podataka se koristi kao ulaz tokom jednog ciklusa procesora (Single Data)
- Izvršavanje je determinističko (predodređeno)
- Najstariji i do skora preovlađujući oblik računara
  - Stariji PC računari, mainframe računari

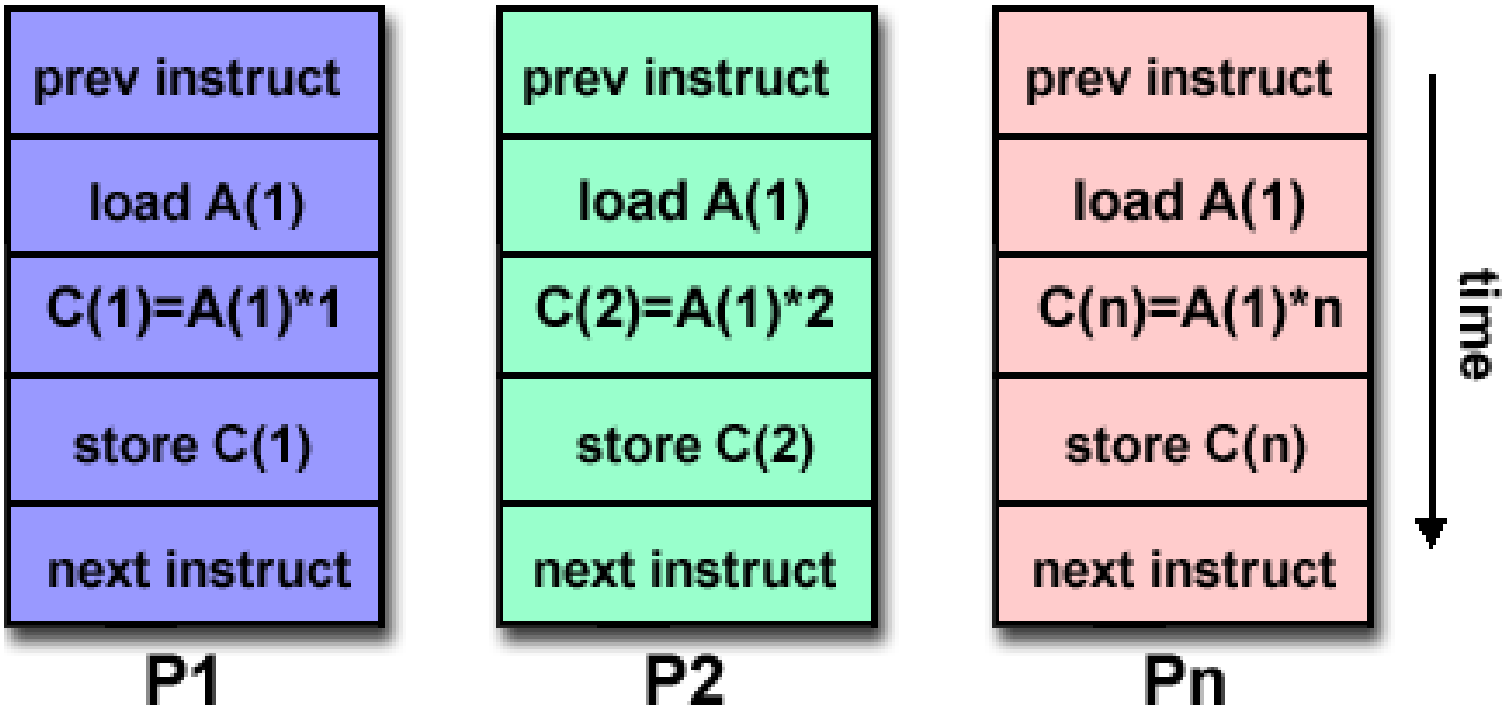
Single Instruction, Multiple Data (SIMD) (1)



Single Instruction, Multiple Data (SIMD) (2)

- Vrsta paralelnog računara
  - Veliki broj jedinica za izvršavanje malog kapaciteta
  - Moderni centralni procesori imaju SIMD jedinice, kao i grafički procesori
- Svi procesori izvršavaju istu instrukciju tokom jednog ciklusa (Single Instruction)
- Svaki procesor može raditi sa različitim podatkom (Multiple Data)
- Sinhrono i determinističko izvršavanje
- Najpogodniji su za probleme koje karakteriše visok stepen regularnosti
  - Obrada slika, obrada audio signala, …

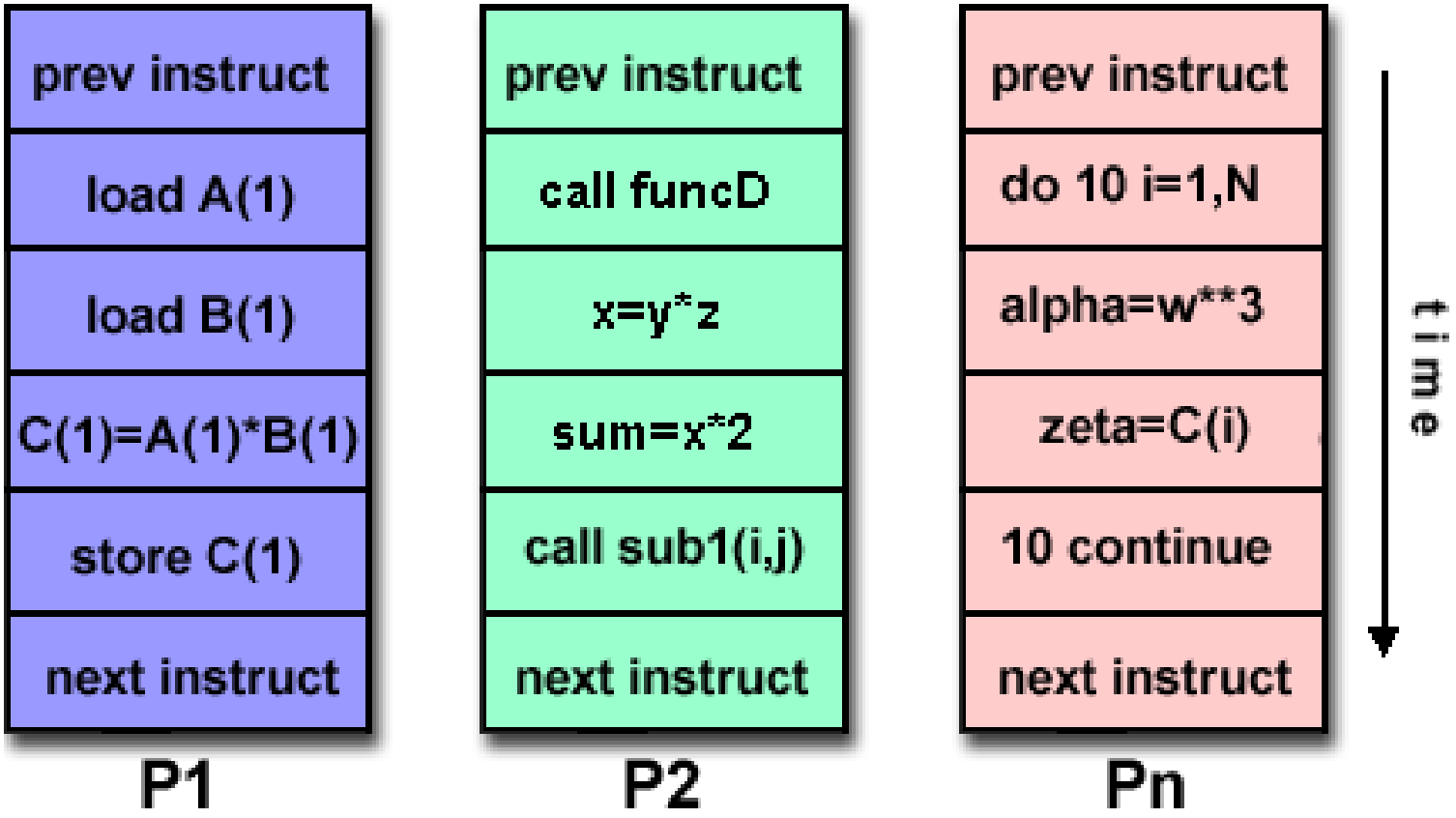
Mulitple Instruction, Single Data (MISD) (1)



Mulitple Instruction, Single Data (MISD) (2)

- Pojedinačni tok podataka se deli na više procesorskih jedinica (Single Data)
- Svaka procesorska jedinica radi nad podacima nezavisno, koristeći nezavisan set instrukcija (Multiple Instruction)
- Malo pravih primera ove klase paralelnih računara
  - Eksperimentalni Carnegie-Mellon C.mmp računar (1971)
- Moguća (uglavnom teorijska) polja primene
  - Višefrekvencijski filtar koji radi sa jednim tokom podataka (pojedinačnim signalom)
  - Razbijanje pojedinačne kodirane poruke korišćenjem više različitih kriptografskih algoritama

Multiple Instruction, Multiple Data (MIMD) (1)



Multiple Instruction, Multiple Data (MIMD) (2)

- Trenutno najzastupljeniji tip paralelnih računara
- Svaki procesor može da izvršava različit set instrukcija (Multiple Instruction)
- Svaki procesor može da radi sa različitim setom podataka (Multiple Data)
- Izvršavanje može biti sinhrono ili asinhrono, determinističko ili nedeterminističko
- Većina današnjih superkompjutera, računarski gridovi, SMP i CMP računari, sadašnji PC računari

Memorijska arhitektura paralelnih računara

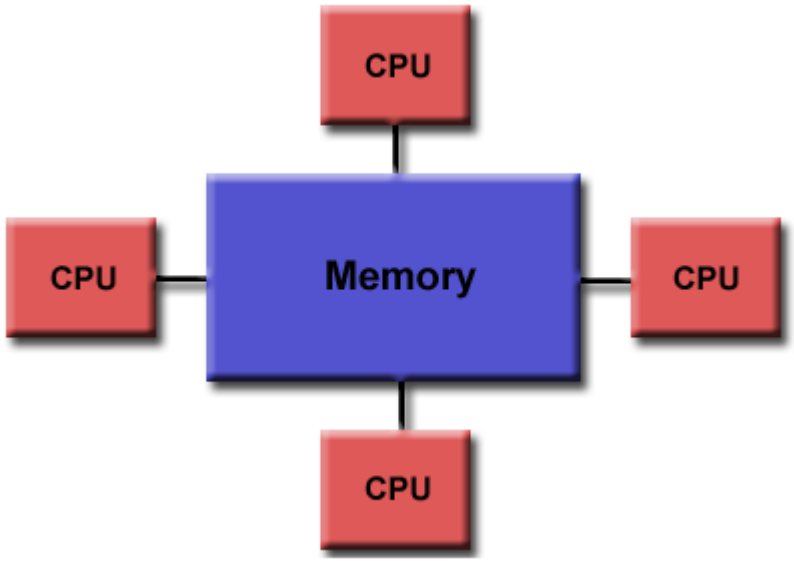
Deljena memorija (1)

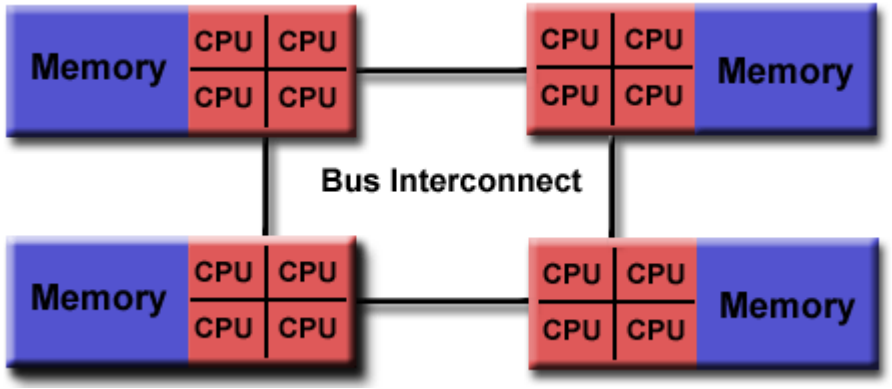
- Svi procesori u multiprocesorskom sistemu imaju mogućnost da pristupaju svim memorijskim resursima kao globalnom adresnom prostoru
- Više procesora radi nezavisno,
  ali koriste iste memorijske resurse
  - Promene u memoriji od strane jednog procesora vidljive su ostalim procesorima
- Multiprocesori sa deljenom memorijom se prema vremenu pristupa memoriji mogu podeliti na dve klase
  - UMA (Uniform Memory Access)
  - NUMA (Non-Uniform Memory Access)

Deljena memorija (2)

- Uniform Memory Access (UMA)
  - Najčešća memorijska arhitektura kod Symmetric Multiprocessor (SMP) mašina
    - Procesori u SMP mašinama su identični
  - Podjednak pristup i vreme pristupa memoriji za sve procesore
  - Najčešće se održava koherencija keš memorija
    - Koherencija keša se održava na hardverskom nivou
- Non-Uniform Memory Access (NUMA)
  - Često se ostvaruje kroz fizičko povezivanje dva ili više SMP sistema
  - Nemaju svi procesori podjednako vreme pristupa svim memorijama u sistemu
    - Pristup memoriji kroz link je sporiji
    - Može se postići koherencija keš memorija (cc-NUMA), ali ona nije obavezna

Deljena memorija (3)





Deljena memorija (4)

- Prednosti deljene memorije
  - Globalni adresni prostor olakšava programiranje ovakvih sistema
  - Deljenje podataka između zadataka je brzo (često i uniformno) zbog blizine memorije procesorima
- Nedostaci deljene memorije
  - Slaba skalabilnost između memorije i broja procesora
    - Povećanje broja procesora u sistemu geometrijski povećava saobraćaj na deljenoj magistrali
    - Kod sistema koji održavaju koherenciju keš memorija značajno se povećava režijski saobraćaj za održavanje koherencije
  - Programer je odgovoran za sinhronizacione primitive za korektan pristup memoriji
  - Cena ovakvih sistema značajno raste sa porastom broja procesora u mašini sa deljenom memorijom

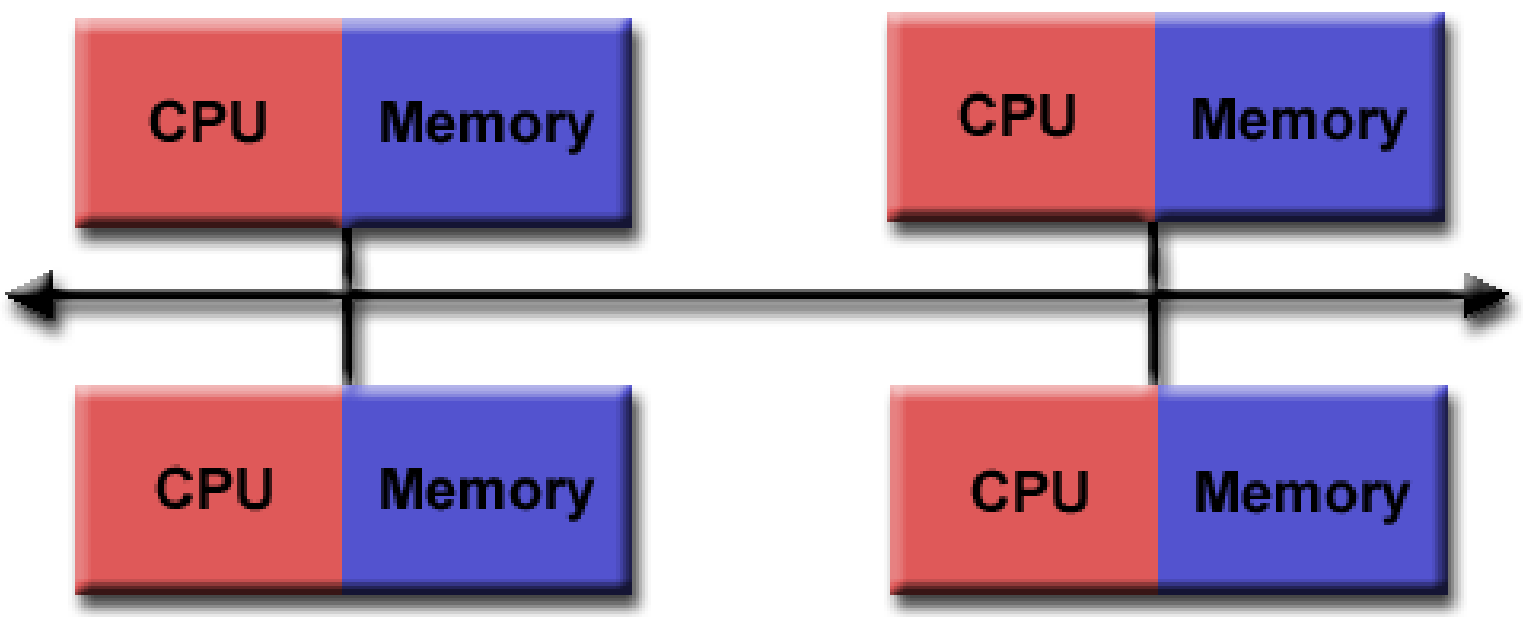
Distribuirana memorija (1)

- Kod sistema sa distribuiranom memorijom postoji interkonekciona komunikaciona mreža koja povezuje procesore i memorije
- Svaki procesor ima svoju lokalnu memoriju koju koristi nezavisno od drugih procesora
  - Memorijske adrese jednog procesora se ne mapiraju kod drugih procesora
  - Ne postoji globalni adresni prostor
  - Promene koje se dese u lokalnoj memoriji jednog procesora ne utiču na memorije drugih procesora
  - Nije potreban koncept koherencije keš memorije

Distribuirana memorija (2)

- Kada je procesoru potreban podatak iz memorije drugog procesora, obaveza je programera da definiše način na koji će i kada će podatak biti dobavljen
  - Sinhronizacija između zadataka je programerova odgovornost
- Interkonekciona mreža
  - Može biti jednostavna (Ethernet, Infiniband, Myrinet)
  - Postoje i namenske interkonekcione mreže poput hiper kocke, različitih *mesh*-eva i sl.

Distribuirana memorija (3)



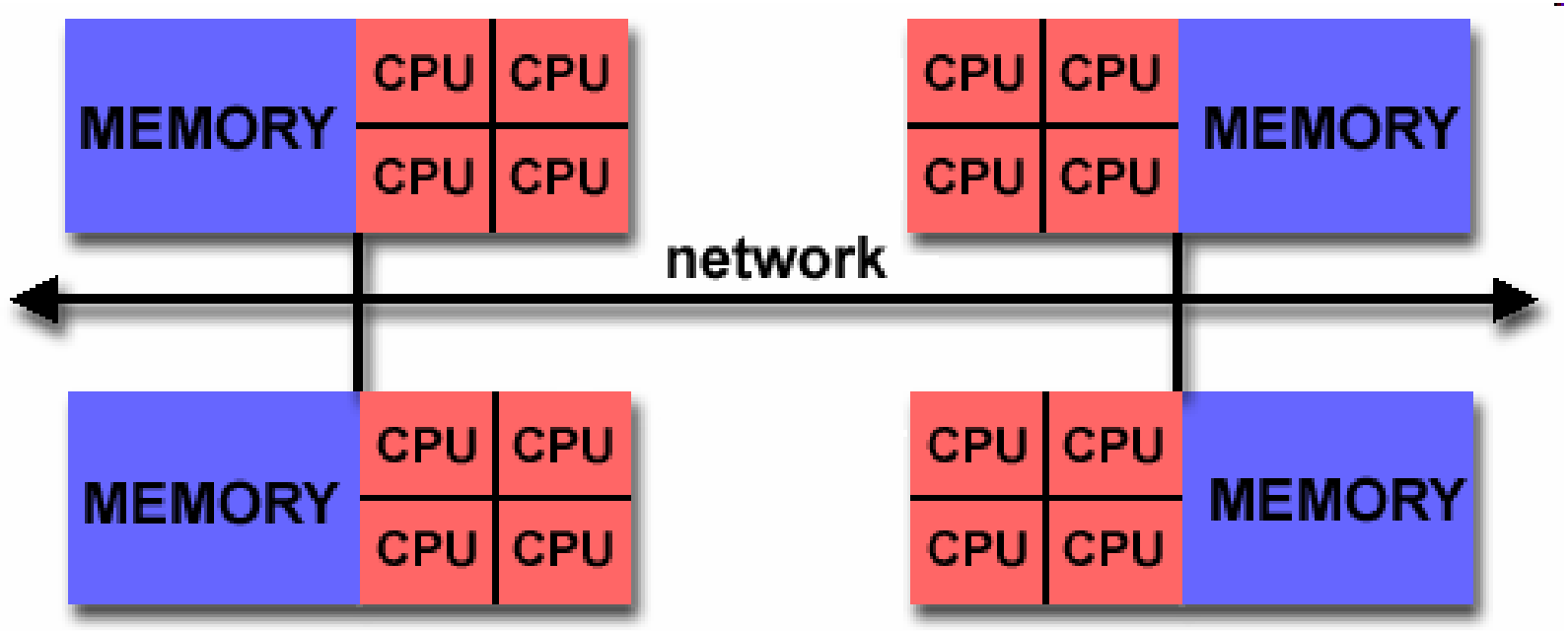
Distribuirana memorija (4)

- Prednosti distribuirane memorije
  - Memorija je skalabilna prema broju procesora
    - Svaki procesor ima svoju memoriju
  - Nema potrebe za održavanjem koherencije keš memorija
  - Niski troškovi primene i eksploatacije
    - Mogu se koristiti *off-the-shelf* komponente i opšte mreže
- Nedostaci distribuirane memorije
  - Programer je odgovoran za celokupnu komunikaciju i razmenu podataka između procesora
  - Može biti teško implementirati strukture podataka zasnovane na globalnoj memoriji na ovakvoj memorijskoj organizaciji
  - Nejednako vreme pristupa različitim memorijama

Hibridni pristup (1)

- Najbrži računari na svetu koriste hibridni pristup
  - Postoji i Shared Memory i Distributed memory komponenta u sistemu (Distributed Shared Memory)
- Više SMP mašina je povezano putem interkonekcione mreže
  - Procesor unutar SMP mašine može pristupiti celokupnoj memoriji svoje mašine
  - Ukoliko je potrebno da se neki podaci prenesu iz memorije jedne SMP mašine na drugu, koristi se mreža
- Hibridni pristup je trenutno preovlađujući

Hibridni pristup (2)



Programski modeli za paralelno programiranje

Pregled

- Paralelni programski model predstavalja apstrakciju iznad hardvera i memorijskih arhitektura
  - Korišćeni programski model najčešće zavisi od problema koji treba rešiti, dostupnosti i ličnog izbora
- Postoji nekoliko paralelnih programskih modela
  - Deljena memorija (*shared memory*)
  - Niti (*threads*)
  - Prosleđivanje poruka (*message passing*)
  - *Data parallel*
  - Hibridni i drugi modeli
- Ovi programski modeli nisu specifični za pojedinačni tip mašine ili memorijske arhitekture
  - Teorijski, svaki od ovih modela se može implementirati na bilo kom hardveru

Model deljene memorije

- Zadaci dele zajednički adresni prostor po kome čitaju i pišu asinhrono
- Različiti mehanizmi kao što su brave i semafori se koriste za kontrolu pristupa deljenoj memoriji
- Ne postoji princip "vlasništva" nad podatkom
  - Komunikacija između zadataka uprošćena, što doprinosi jednostavnijem razvoju programa
- Problem održavanja lokalnosti podataka
  - Teško razumevanje i složeno upravljanje razmeštajem podataka
  - Vrlo složeno za prosečnog korisnika (tj. programera)

Niti (1)

- Prema ovom modelu paralelnog programiranja, proces može imati više konkurentnih grana izvršavanja
  - Jednostavnije od modela deljene memorije
- Niti su često zastupljene u operativnim sistemima i sistemima sa deljenom memorijom
- Implementacije se obično sastoje od
  - Biblioteka rutina koje se pozivaju iz paralelnog koda
  - Skupa direktiva prevodiocu (ili pretprocesoru) ugrađenih u sekvencijalni ili paralelni izvorni kod
- U oba slučaja programer je zadužen za paralelizaciju
- Niti su uobičajen koncept u računarstvu
  - Mnogo nekompatibilnih standarda i platformi

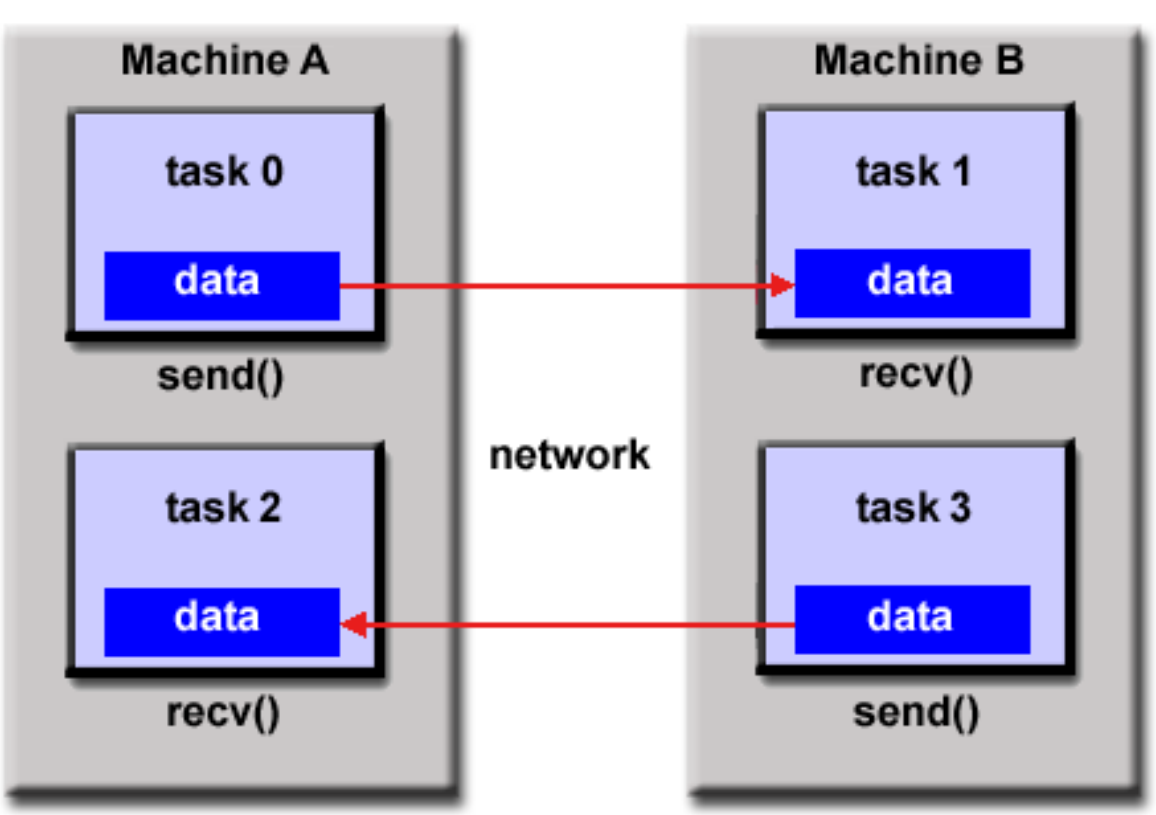
Niti (2)

- POSIX niti (Pthreads)
  - Biblioteke rutina koje se pozivaju iz paralelnog koda
  - IEEE POSIX 1003.1c standard (1995-2013)
  - Implementacije samo za jezik C
  - Vrlo eksplicitan paralelizam,
    programer mora da obrati pažnju na detalje
- OpenMP standard
  - Zasnovan na direktivama prevodiocu (pretprocesoru)
  - Može se koristiti na uobičajenom (sekvencijalnom) kodu
  - Fortran i C/C++ implementacije
  - Portabilan i multiplatformski (uključuje Unix/Linux i Windows)
  - Lak za korišćenje
- Microsoft je razvio sopstvenu implementaciju niti, nevezanu za pomenute standarde

Prosleđivanje poruka (1)

- Skup zadataka koristi svoje lokalne memorije prilikom izračunavanja
- Više zadataka može postojati bilo na istoj fizičkoj mašini, bilo na određenom broju mašina
- Zadaci razmenjuju podatke komunicirajući putem slanja i primanja poruka
- Razmena podataka obično uključuje saradnju između procesa koji učestvuju u komunikaciji
  - Operacija slanja se mora upariti sa odgovarajućom operacijom za prijem

Prosleđivanje poruka (2)



Prosleđivanje poruka (3)

- Implementacije
  - Ovaj model se obično implementira kroz biblioteku rutina koje programer poziva iz izvornog koda
  - Programer je odgovoran za paralelizaciju problema
- Message Passing Interface (MPI)
  - MPI-1 (1994), MPI-2 (1996) i MPI-3 (2012)
  - Trenutno je industrijski standard za razmenu poruka
  - Kod arhitektura sa deljenom memorijom, MPI implementacije obično ne koriste mrežu za komunikaciju između zadataka, već koriste deljenu memoriju zbog poboljšanja performansi

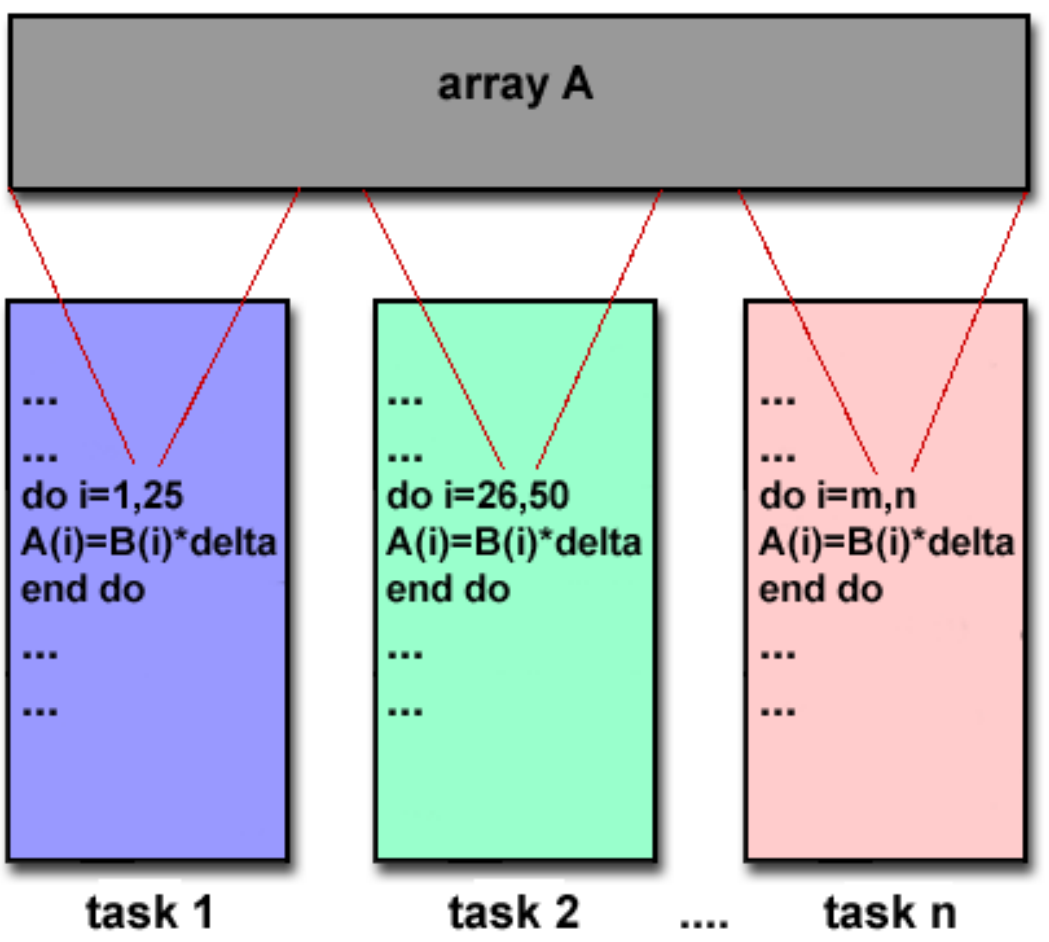
Data parallel model (1)

- Ponekad se naziva Partitioned Global Address Space (PGAS)
  - Adresni prostor je globalno vidljiv
- Paralelizacija se vrši nad skupom podataka
  - Podaci su tipično organizovani u pravilnu strukturu, kao što su niz i kocka
- Skup zadataka zajednički radi na istoj strukturi podataka
  - Svaki zadatak radi na različitom delu iste strukture
  - Zadaci rade istu operaciju na njihovom delu posla
  - Na sistemima sa deljenom memorijom, svi zadaci imaju pristup celoj strukturi kroz globalnu memoriju
  - Na sistemima sa distribuiranom memorijom se struktura deli i postoji u delovima u lokalnoj memoriji svakog zadatka

Data parallel model (2)

- Implementira se obično zadavanjem data-parallel instrukcija
  - Ostvarenje ili kroz biblioteke potprograma ili kroz direktive data parallel kompajlerima
- Često eksperimentalna rešenja
  - Coarray Fortran 90/95, High Performance Fortran (HPF) i sl.
  - Unified Parallel C (UPC)
  - Global Arrays biblioteka za distribuirane sisteme
  - Chapel jezik za Cray superračunare

Data parallel model (3)



Drugi modeli (1)

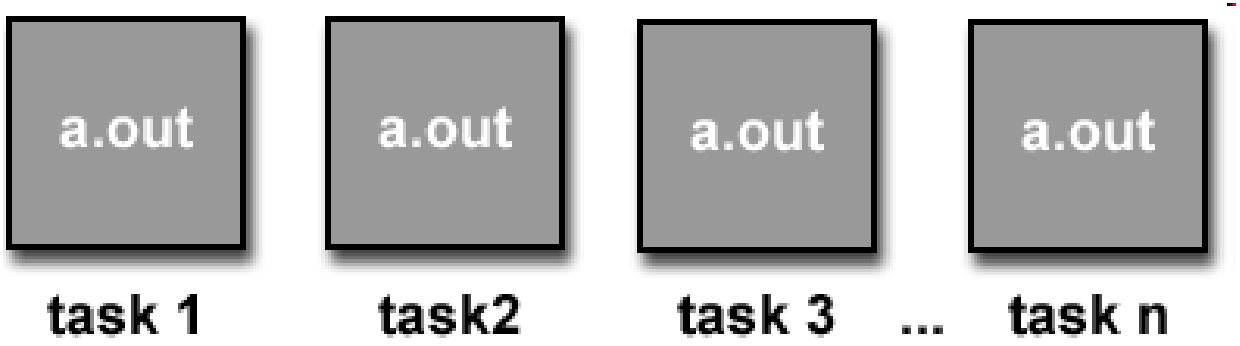
- Hibridni model
  - Bilo koja dva (ili više) modela se kombinuju
  - Česta je kombinacija MPI i POSIX/OpenMP za hibridni model memorijske arhitekture (DSM)
  - Sreće se i kombinacija data parallel i MP modela
- Single Program Multiple Data (SPMD)
  - Pojedinačan program se izvršava od strane više zadataka simultano
  - U svakom trenutku zadaci mogu izvršavati iste ili različite instrukcije unutar programa
  - Pojedinačan zadatak može izvršavati samo deo programa, u zavisnosti od ugrađene programske logike
  - Svi zadaci mogu koristiti različite podatke

Drugi modeli (2)

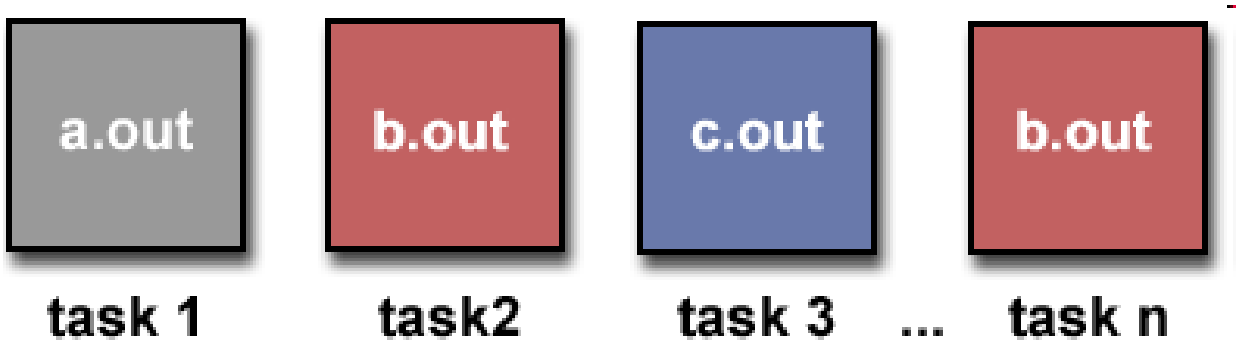
- Multiple Program Multiple Data (MPMD)
  - Ovakve aplikacije imaju više programa
  - Dok se aplikacija izvršava u paraleli,
    svaki zadatak može izvršavati isti ili drugačiji program
  - Svi zadaci mogu koristiti različite podatke
- SPMD i MPMD su programski modeli visokog nivoa
  - Mogu se izgraditi na bilo kojoj kombinaciji
    pomenutih paralelnih programskih modela
- GPGPU (General Purpose computation on GPU)
  - Upotreba grafičkog procesora (GPU) za računanje
  - Compute Unified Device Architecture (CUDA), OpenCL

Drugi modeli (3)

SPMD



MPMD



Razvoj paralelnih programa

Automatska i manuelna paralelizacija (1)

- Dizajn i implementacija paralelnih programa je tipično manuelni, iterativni proces
  - Programer je odgovoran i za prepoznavanje i za implementaciju paralelizma u nekom programu
  - Često je to vremenski zahtevan i složen iterativni proces, podložan greškama
- Vremenom su razvijeni različiti alati koji pomažu prevođenje sekvencijalnog u paralelni kod
  - Najčešće su to prevodioci ili pretprocesori prilagođeni da rade paralelizaciju koda

Automatska i manuelna paralelizacija (2)

- Potpuno automatska paralelizacija
  - Prevodilac sam analizira izvorni kod i pronalazi delove pogodne za paralelizaciju
    - Petlje su najpogodnije za paralelizaciju
  - Ne daje uvek željene rezultate
    - Može dovesti do pogrešnih rezultata prilikom korišćenja paralelizovanog programa
    - Performanse mogu da opadnu
    - Nedostatak fleksibilnosti
- Paralelizacija upravljana od strane programera
  - Direktive paralelnom kompajleru kako da paralelizuje određeni deo koda
    - Primer je OpenMP

Razumevanje problema koji se rešava (1)

- Prvi korak u razvoju paralelnog softvera je razumevanje problema koji treba rešiti u paraleli
  - Ukoliko se polazi od postojećeg sekvencijalnog koda, potrebno je i njega dobro razumeti
  - Potrebno je utvrditi da li je problem uopšte moguće paralelizovati
- Razmotriti druge algoritme za rešavanje istog problema
  - Često su restrukturirani algoritmi pogodniji za paralelizaciju

Razumevanje problema koji se rešava (2)

- Potrebno je paralelizovati najbitnije delove (*hotspots*)
  - Delovi koda koji troše najviše vremena
  - Nema svrhe paralelizovati delove koda koji malo koriste procesor
- Identifikovati inhibitore paralelizma (*bottlenecks*)
  - Uobičajeno su to I/O operacije i komunikacija
  - Vrlo često su to zavisnosti po podacima
- Profajleri koda i programi za analizu performansi mogu pomoći pri određivanju navedenih mesta

Razumevanje problema koji se rešava (3)

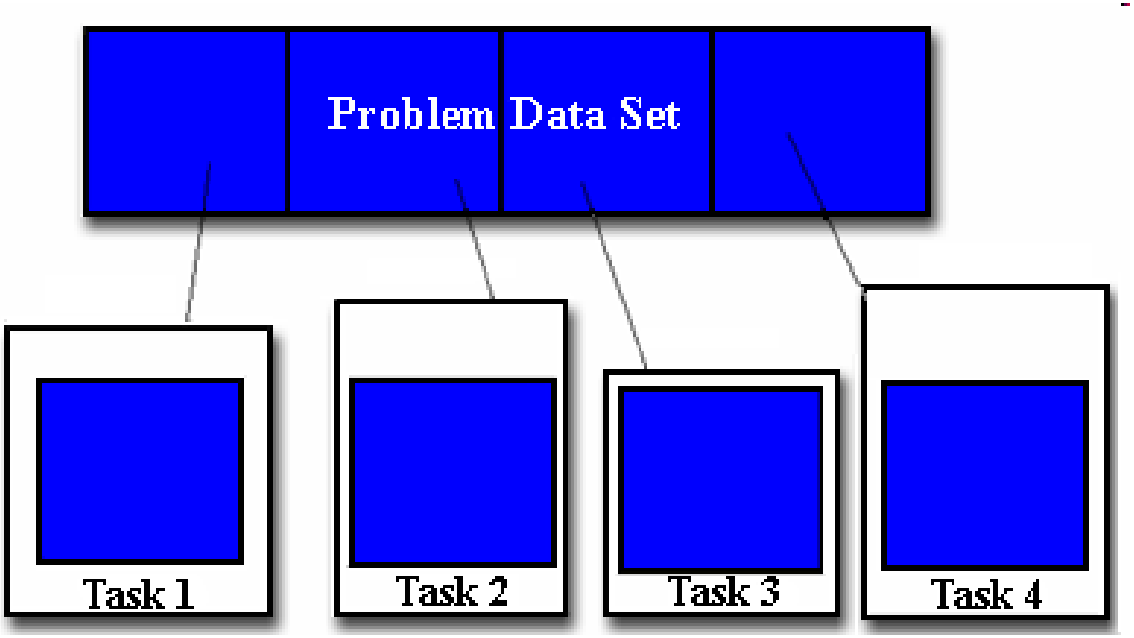
- Primeri:
  - Množenje dve matrice
    - Problem izuzetno pogodan za paralelizaciju
    - Računanje jednog elementa rezultujuće matrice je nezavisno u odnosu na druge
  - Izračunavanje Fibonačijevih brojeva
    - Problem nemoguć za paralelizaciju
    - Izračunavanje (k+2)-og broja u seriji zahteva poznavanje (k+1)-og i k-tog Fibonačijevog broja

Dekompozicija problema

- Jedan od prvih koraka prilikom dizajniranja paralelnih programa je podela (particionisanje) problema koji se rešava na diskretne delove posla
  - Diskretni delovi se potom šalju različitim paralelnim zadacima na obradu
- Dva tipa dekompozicije
  - Dekompozicija domena
  - Funkcionalna dekompozicija

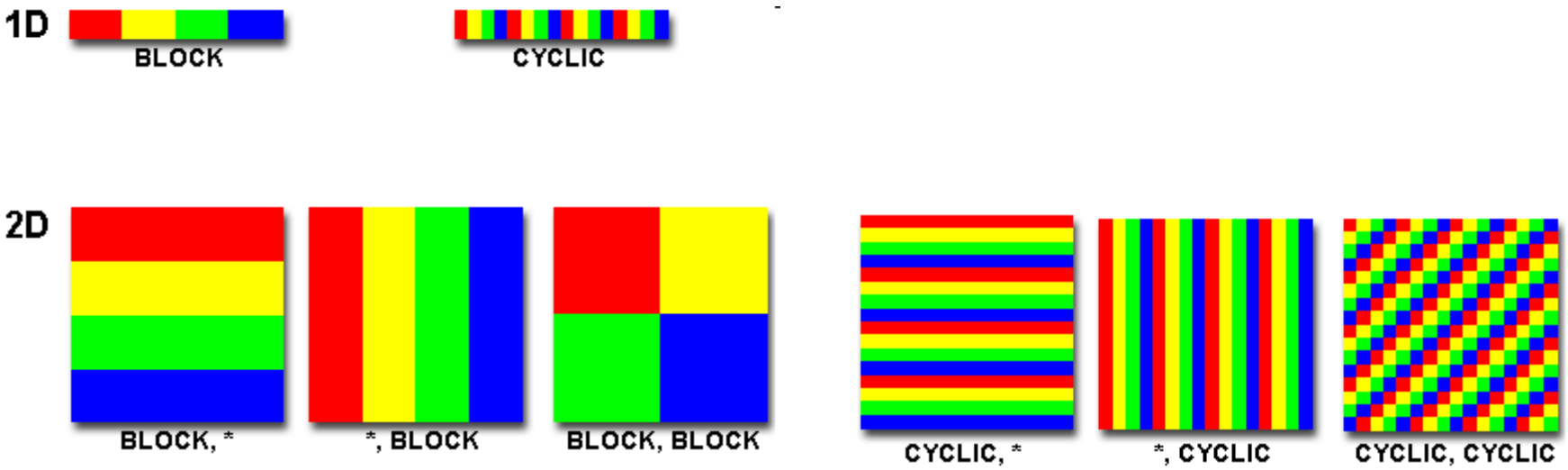
Dekompozicija domena (1)

- Podaci u vezi sa problemom se dele
  - Svaki paralelni zadatak potom radi na delu podataka



Dekompozicija domena (2)

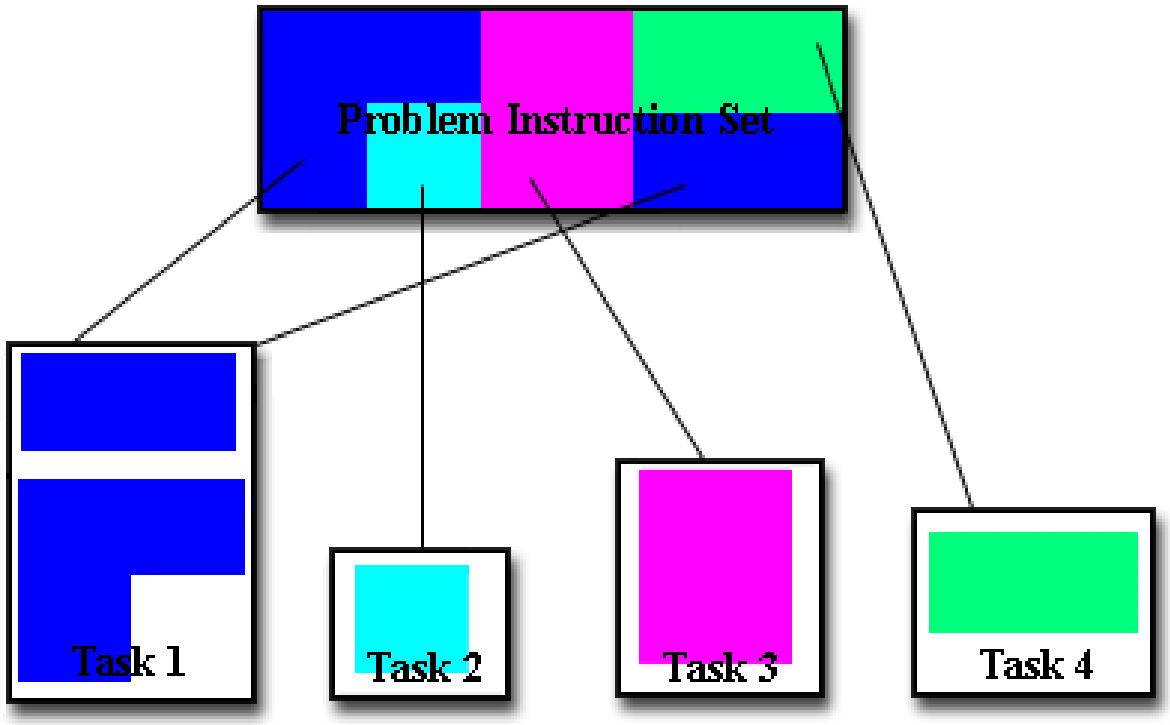
- Nekoliko načina za podelu podataka
  - 1D, 2D, ciklično



Funkcionalna dekompozicija (1)

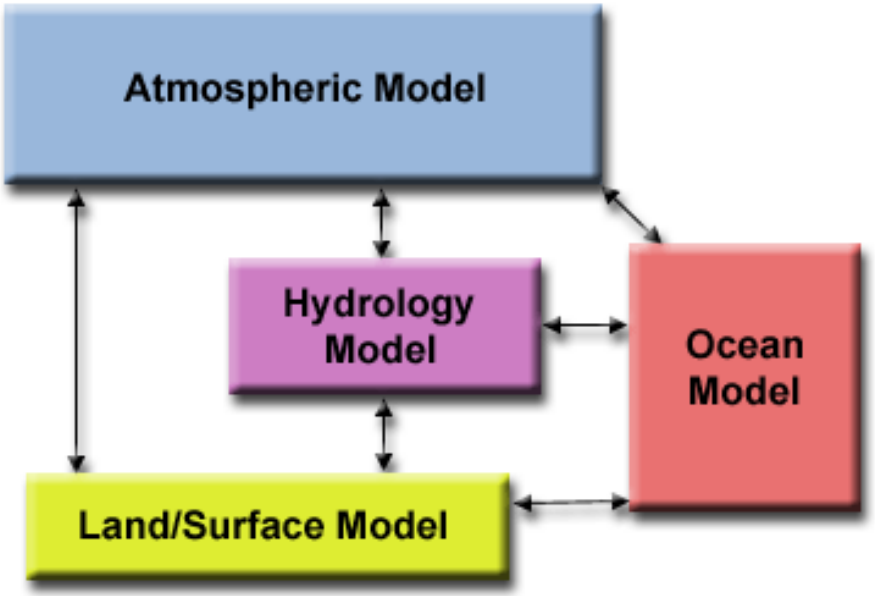
- Svaki zadatak dobija deo problema koji treba da se reši
  - Fokus je na računskom poslu koji treba izvršiti, a manje nad podacima koji se koriste prilikom računanja
- Funkcionalna dekompozicija je pogodna za određene klase problema
  - Modelovanje ekosistema
  - Procesiranje signala
  - Modelovanje klimatskih promena

Funkcionalna dekompozicija (2)



Funkcionalna dekompozicija (3)

- Primer – modeliranje klimatskih promena
  - Svaka komponenta modela je zaseban zadatak
  - Komponente međusobno komuniciraju da dođu do neophodnih podataka



Komunikacija između zadataka (1)

- Potreba za komunikacijom između zadataka zavisi od tipa problema koji se rešava
  - Ne postoji potreba za komunikacijom
    - Ovakvi problemi se nazivaju "*embarrassingly parallel*", jer zahtevaju vrlo malo komunikacije između zadataka
    - Primeri su određeni algoritmi za obradu slike
  - Postoji potreba za komunikacijom
    - Većina paralelnih aplikacija zahteva razmenu podataka između paralelnih zadataka
    - Simulacija klimatskih promena je takav primer
      - Neophodna je komunikacija između različitih delova modela

Komunikacija između zadataka (2)

- Komunikacija među zadacima iziskuje dodatne režijske troškove
  - Glavni izvor paralelnog "overhead-a"
  - Zbog sinhronizacije, zagušenja komunikacionih kanala, čekanja
- Kašnjenje (*latency*) i propusni opseg (*bandwidth*) utiču na performanse komunikacije
  - Slanje velikog broja malih poruka dovodi do toga da kašnjenje dominira u režijskim troškovima komunikacije
- Vidljivost komunikacije je bitan faktor
  - Da li je komunikacija eksplicitna ili implicitna?

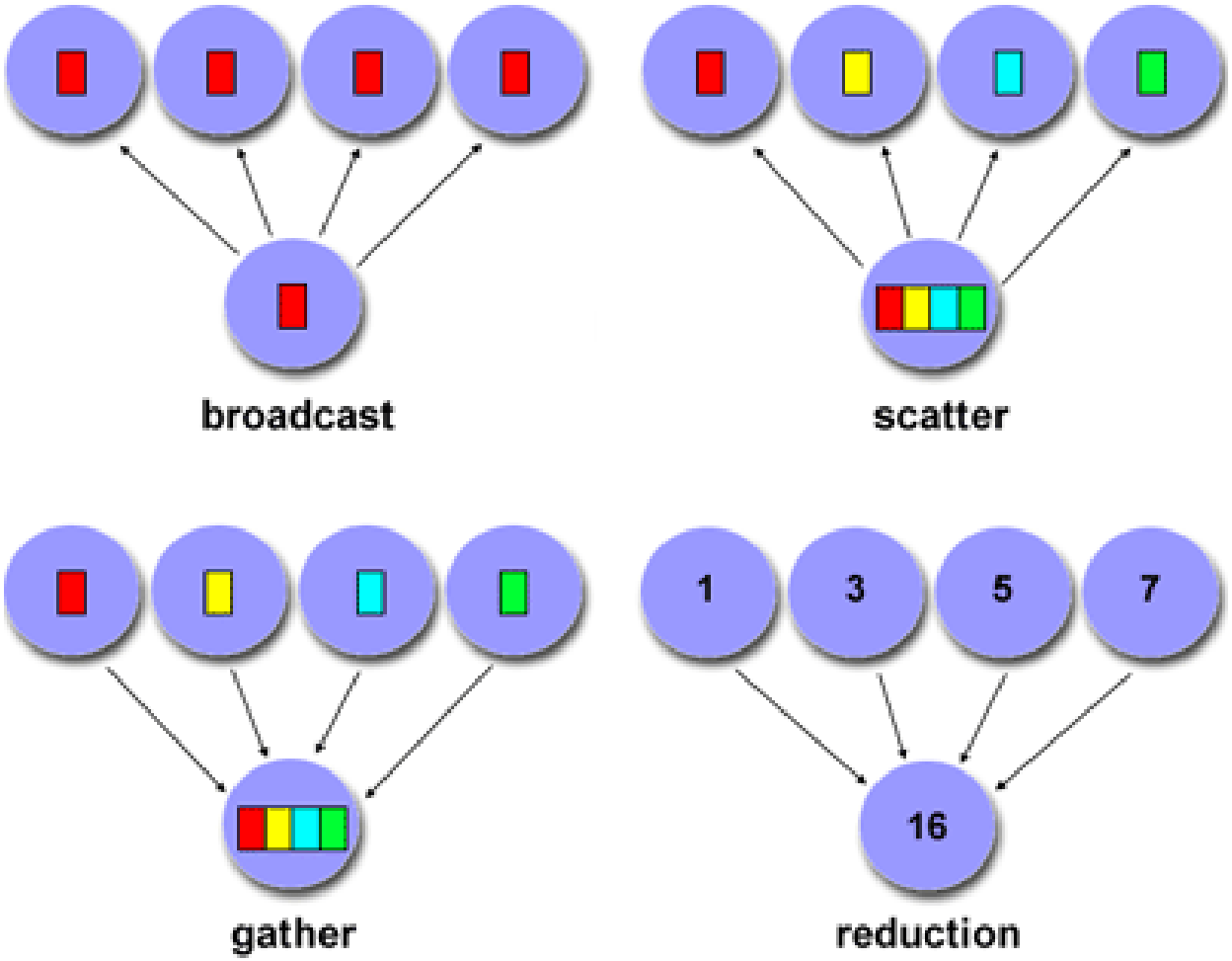
Komunikacija između zadataka (3)

- Sinhrona i asinhrona komunikacija
  - Sinhrona komunikacija zahteva neku vrstu "handshaking" protokola između zadataka
    - Može je implementirati programer ili može biti transparentna
    - Sinhrona komunikacija je blokirajuća, jer ostali posao mora da sačeka dok se komunikacija ne završi
  - Asinhrona komunikacija dozvoljava prenos podataka između zadataka nezavisno jedan od drugih
    - Trenutak kada primalac stvarno prima podatke nije bitan, pošto pošiljalac može da nastavi da radi drugi posao
    - Asinhrona komunikacija je neblokirajuća, jer pošiljalac može da nastavi da radi svoj posao
  - Da li je dobitak od preklapanja komunikacije i računanja vredan komplikovanja programskog koda?

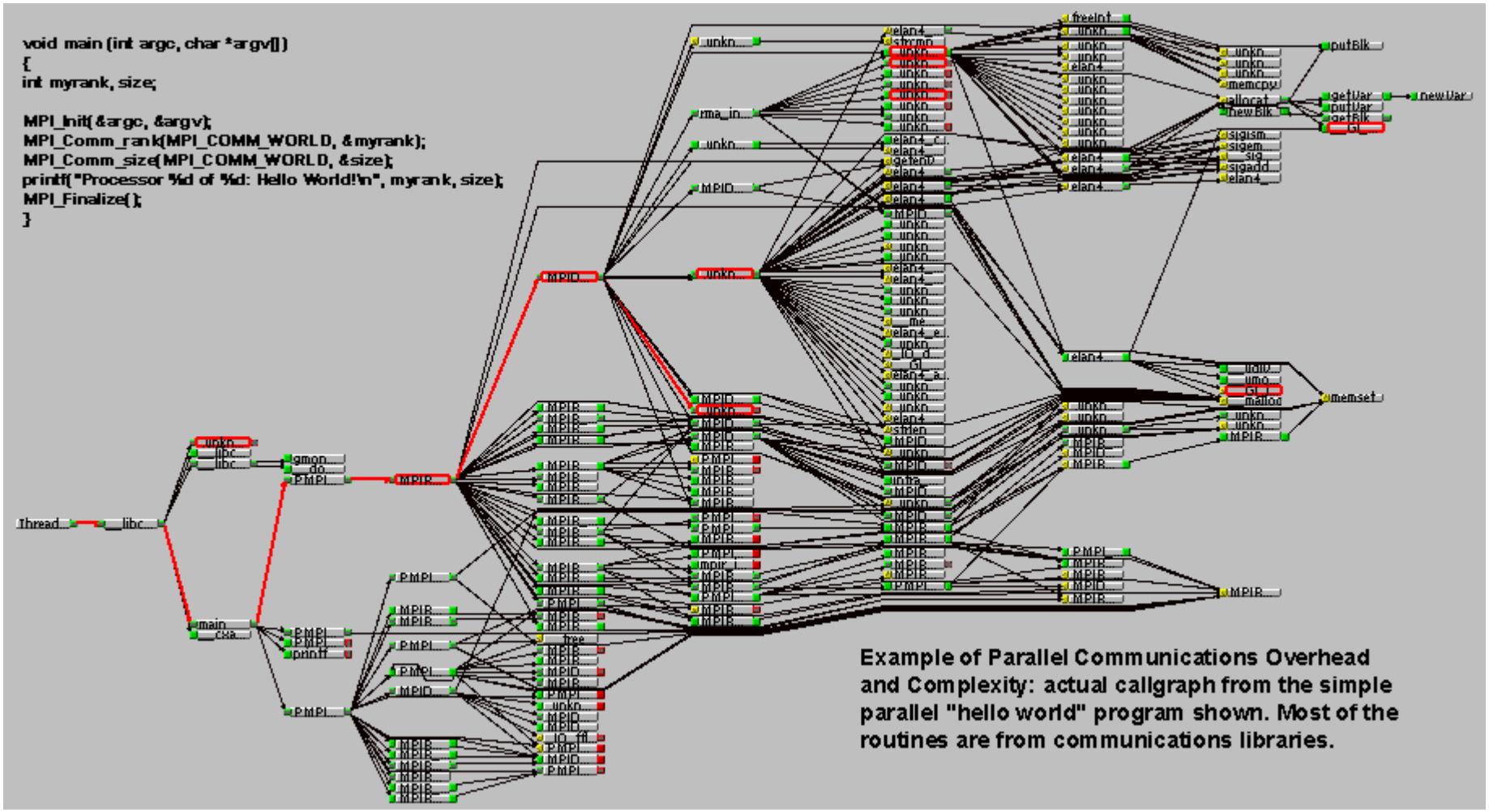
Komunikacija između zadataka (4)

- Opseg komunikacije
  - Pojedinačna, između dva zadatka (*point-to-point*)
    - Uključuje komunikaciju dva zadatka, od kojih se prvi ponaša kao proizvođač/pošiljilac, a drugi kao potrošač/primalac podataka
  - Grupna, između više zadataka (*collective*)
    - Uključuje komunikaciju više zadataka, najčešće pripadnika iste grupe
- Efikasnost komunikacije
  - Performanse istog modela variraju između platformi
  - Zavisi i od vrste mreže za komunikaciju
    - Koju mrežu izabrati ukoliko ima više dostupnih?

Primeri kolektivne komunikacije



Režijsko vreme i kompleksnost



Sinhronizacija između zadataka (1)

- Sinhronizacija je neizostavna za koordinaciju paralelnih zadataka
- Sinhronizacija na barijeri (*barrier*)
  - Obično uključuje sve zadatke
  - Svaki zadatak radi posao dok ne dođe do barijere, kada prekida posao i blokira se
  - Kada poslednji zadatak dosegne barijeru svi zadaci su sinhronizovani
    - Barijeru obično sledi kritična sekcija koja se mora sekvencijalno izvršiti
    - Alternativno, zadaci se deblokiraju i nastavljaju svoj posao

Sinhronizacija između zadataka (2)

- Brava (*lock*)
  - Može učestvovati bilo koji broj zadataka
  - Samo jedan zadatak može istovremeno da koristi bravu i uđe u kritičnu sekciju
  - Prvi zadatak koji naiđe na slobodnu bravu je zaključava i obavlja posao, ostali moraju da čekaju
- Semafori (*semaphores*)
  - Može učestvovati bilo koji broj zadataka
  - Određeni broj zadataka može istovremeno da koristi semafor i koristi resurs ili uđe u kritičnu sekciju
  - Blokirajući, binarni (brave) i brojački semafori

Sinhronizacija između zadataka (3)

- Uslovne promenljive (*conditional variables*)
  - Naredni zadatak nastavlja tek kad prethodni završi
  - Postoji red zadataka koji čekaju na uslovnoj promenljivoj
    - zadatak se obavezno blokira prilikom izvršavanja operacije čekanja na uslovnoj promenljivoj
- Sinhrone komunikacione primitive
  - Uključuje samo one zadatke koji komuniciraju
  - Zahteva određeni nivo koordinacije između zadataka koji komuniciraju
    - Pre nego što izvrši slanje, pošiljalac mora da dobije potvrdu da je primalac spreman da prihvati poruku

Zavisnosti po podacima (1)

- Zavisnost u programu postoji kada redosled izvršavanja naredbi u programu utiče na krajnji rezultat određenog dela programa ili celog programa
  - Zavisnost između instrukcija postoji ako će izmena redosleda instrukcija uticati na rezultate programa
  - Zavisnost po podacima nastaje usled korišćenja memorijskih lokacija od strane više paralelnih zadataka
- Zavisnosti su primarni inhibitori paralelizma!
  - Najčešće rešenje je sinhronizacija zadataka
  - Na sistemima sa distribuiranom memorijom je neophodne podatke potrebno razmeniti na sinhronizacionim tačkama
  - Na sistemima sa deljenom memorijom je potrebno je sinhronizovati operacije čitanja i upisa podataka između paralelnih zadataka

Zavisnosti po podacima (2)

- Primer *loop-dependent* zavisnosti po podacima
  - DO 500 J = MYSTART,MYEND
    A(J) = A(J-1) * 2.0
    500 CONTINUE
- Vrednost A(J-1) se mora izračunati pre vrednosti A(J) i stoga postoji zavisnost po podacima
  - Paralelizam je onemogućen
- *Loop-dependent* zavisnosti je jako bitno uočiti prilikom razvoja paralelnih programa, jer to utiče na paralelizaciju petlji, koja se najčešće vrši u procesu paralelizacije

Zavisnosti po podacima (3)

- Primer *loop-independent* zavisnosti po podacima
  - task 1 task 2

    ------ ------

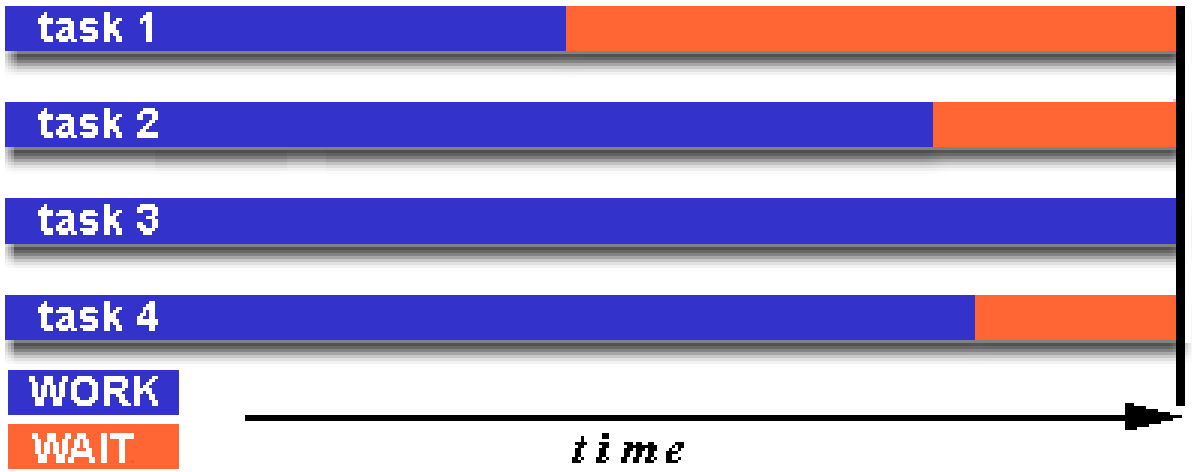
    X = 2 X = 4

    . . . .

    Y = X**2 Y = X**3
- Paralelizam je onemogućen, jer izračunavanje vrednosti Y zahteva poznavanje vrednosti X, koja zavisi od toga da li će vrednost X biti slata između zadataka i koji će je zadatak prvi menjati

Balansiranje opterećenja (1)

- Balansiranje opterećenje (*load balancing*) se odnosi na podelu posla među zadacima tako da svi podaci budu zauzeti sve vreme
  - Osnovni cilj je da svi zadaci stalno nešto rade
- Primer barijere – svi zavise od najsporijeg zadatka

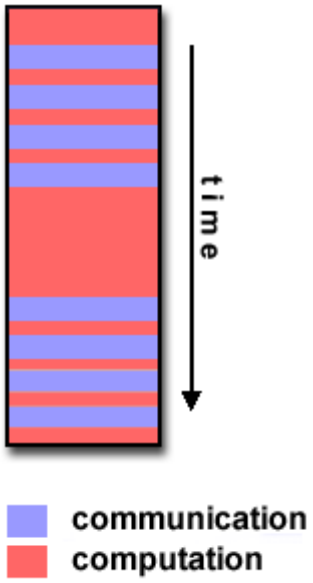


Balansiranje opterećenja (2)

- Ravnomerna podela posla svim zadacima
  - Podela iteracija petlji, ravnomerna raspodela podataka
  - Statička i dinamička podela posla
- Dinamička podela posla
  - Neke vrste problema izazivaju neravnomerno opterećenje i kad su podaci podjednako podeljeni po zadacima
    - Retko posednuti nizovi i retko posednute matrice
    - Adaptivne tehnike obrade, kada određeni zadaci imaju potrebu da izvrše finiju obradu nad određenim podacima
  - Veličina posla se ne može unapred predvideti
    - Korišćenje raspoređivača i depoa poslova može rešiti problem (*scheduler - task pool*)
    - Kada jedan zadatak završi deo posla, staje u red i čeka da dobije sledeći

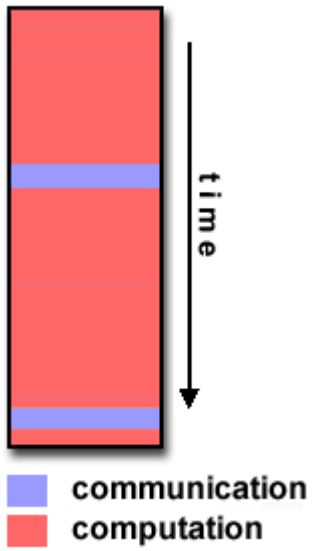
Granularnost (1)

- Odnos vremena potrošenog na računanje i vremena potrošenog na komunikaciju *(computation to communication ratio)*
  - Periodi u kojima se vrši neki računski posao su odvojeni od perioda kada se vrši komunikacija sinhronizacionim događajima
- *Fine-grain* paralelizam
  - Relativno mala količina računskog posla se izvrši između dve komunikacije
  - Olakšava balansiranje opterećenja
  - Veliko režijsko vreme za komunikaciju
    - Možda čak duže od vremena potrebnog za računski posao
  - Smanjena mogućnost za popravljanje performansi



Granularnost (2)

- *Coarse-grain* paralelizam
  - Relativno velika količina računskog posla se izvrši između dva komunikaciona događaja
  - Više mogućnosti za popravljanje performansi
  - Teže je održati balansiranje opterećenja
  - Bolji ukoliko je režijsko vreme komunikacije i sinhronizacije veliko u odnosu na vreme izvršavanja
- Izbor najviše zavisi od konkretnog problema i platforme



I/O operacije

- Ulazno-izlazne operacije su često inhibitor paralelizma
- Moguća rešenja
  - Neke platforme nude paralelne fajl sisteme
    - GPFS, Lustre, OrangeFS/PVFS
    - Podrška u okviru MPI-2 standarda
    - Nedovoljna zrelost
  - Smanjiti I/O poslove na minimum
  - Korišćenje većih blokova podataka za upis
  - Izbegavanje rada sa malim fajlovima
  - Lokalizacija I/O poslova u manji broj zadataka

Analiza performansi paralelnih programa

- Značajno komplikovanija nego kod sekvencijalnih programa
  - Slično je i sa *debugging*-om
- Analizu performansi je potrebno izvršiti i pre i posle paralelizacije programa
  - Bitno je odrediti odnos vremena računanja i vremena komunikacije
- Postoji veliki broj alata za ovu namenu
  - Različiti tajmeri, profajleri i namenski alati
  - Neki su multiplatformski

Performanse

- Tipična metrika:
  - Vreme odgovora (*response time*)
  - Propusni opseg (*throughput*)
- Ubrzanje (*speedup*)
- Vreme izvršavanja
  - "Zidno" (*wall-clock*) vreme:
    - Uključuje sve sistemske *overhead*-e
  - CPU vreme: samo vreme izračunavanja
- Ispitne (*benchmark*) aplikacije
  - Kerneli iz realnih aplikacija (npr., množenje matrica)
  - Mali programi (npr., algoritam za sortiranje)
  - Sintetički benčmarci (npr., *Dhrystone*)
  - Skupovi benčmark aplikacija
    - SPEC, TPC-C, SPLASH, NPB, Parboil, Rodinia, PolyBench…

Ubrzanje

- Ubrzanje (p procesora) = (*speedup*)

$$\frac{\textit{Performanse (p procesora)}}{\textit{Performanse (1 procesor)}}$$

- Za problem fiksne veličine (ulazni podaci), performanse = 1/vreme
- Ubrzanje (p procesora) =

$$\frac{\textit{Vreme (1 procesor)}}{\textit{Vreme(p procesora)}}$$

- Koliko je ubrzanje moguće?

Amdahl-ov zakon (1)

- α – sekvencijalni deo aplikacije, p – broj procesora
- T_i – vreme izvršavanja sa *i* procesora

$$T_p = \alpha \cdot T_1 + \frac{(1-\alpha)\cdot T_1}{p}$$

$$T_1 = \alpha \cdot T_1 + \frac{(1-\alpha)\cdot T_1}{1} = \alpha \cdot T_1 + T_1 - \alpha \cdot T_1 = T_1$$

$$\lim_{p\to\infty} T_p = \alpha \cdot T_1$$

- Ubrzanje je ograničeno sekvencijalnim delom aplikacije nezavisno od broja procesora

Amdahl-ov zakon (2)

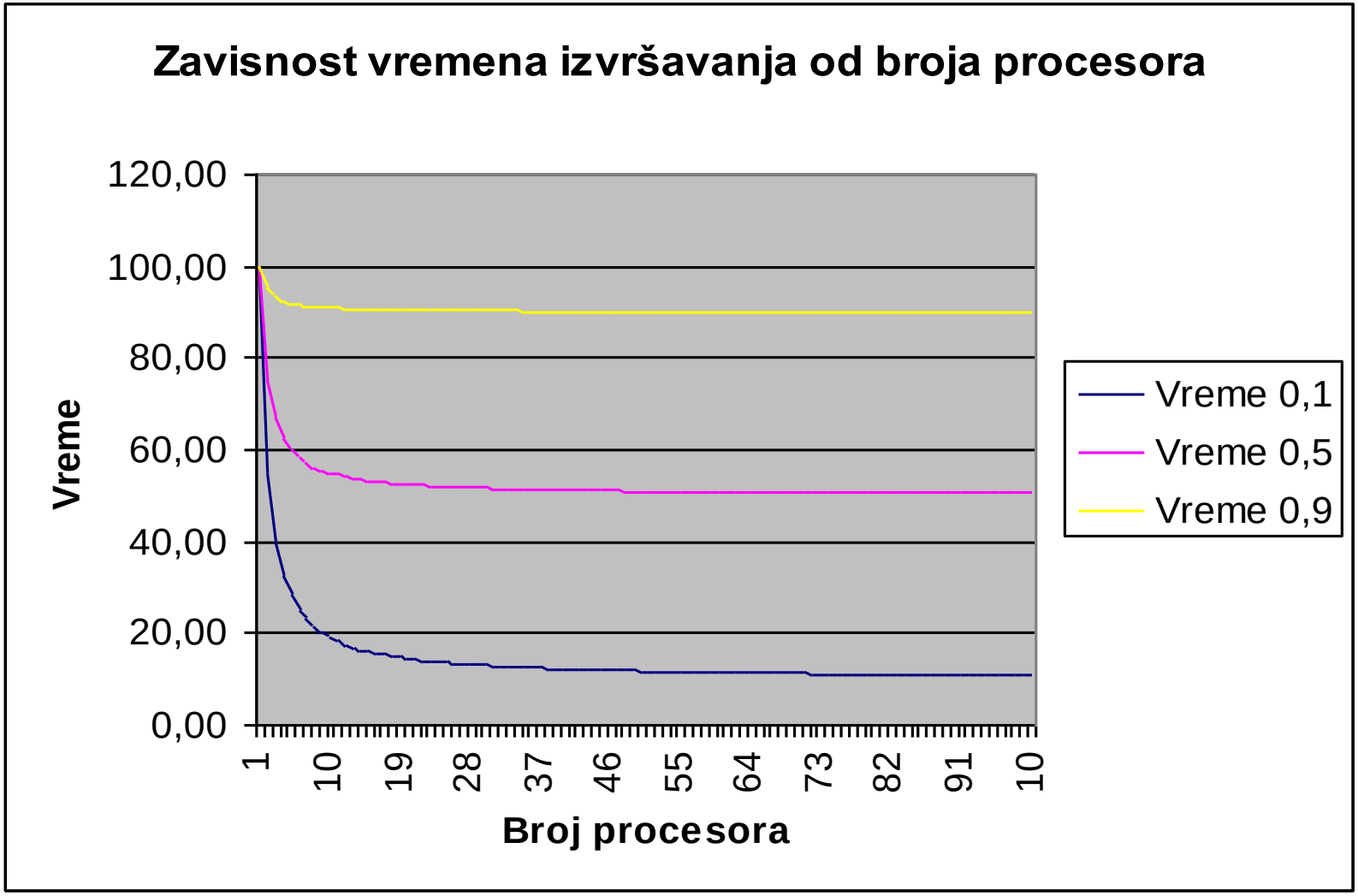
- α – sekvencijalni deo aplikacije, p – broj procesora
- T_i – vreme izvršavanja sa *i* procesora

$$S_p = \frac{p}{1+(p-1)\cdot\alpha} \qquad S_1 = \frac{p}{1+(p-1)\cdot\alpha} = \frac{1}{1+(1-1)\cdot\alpha}$$

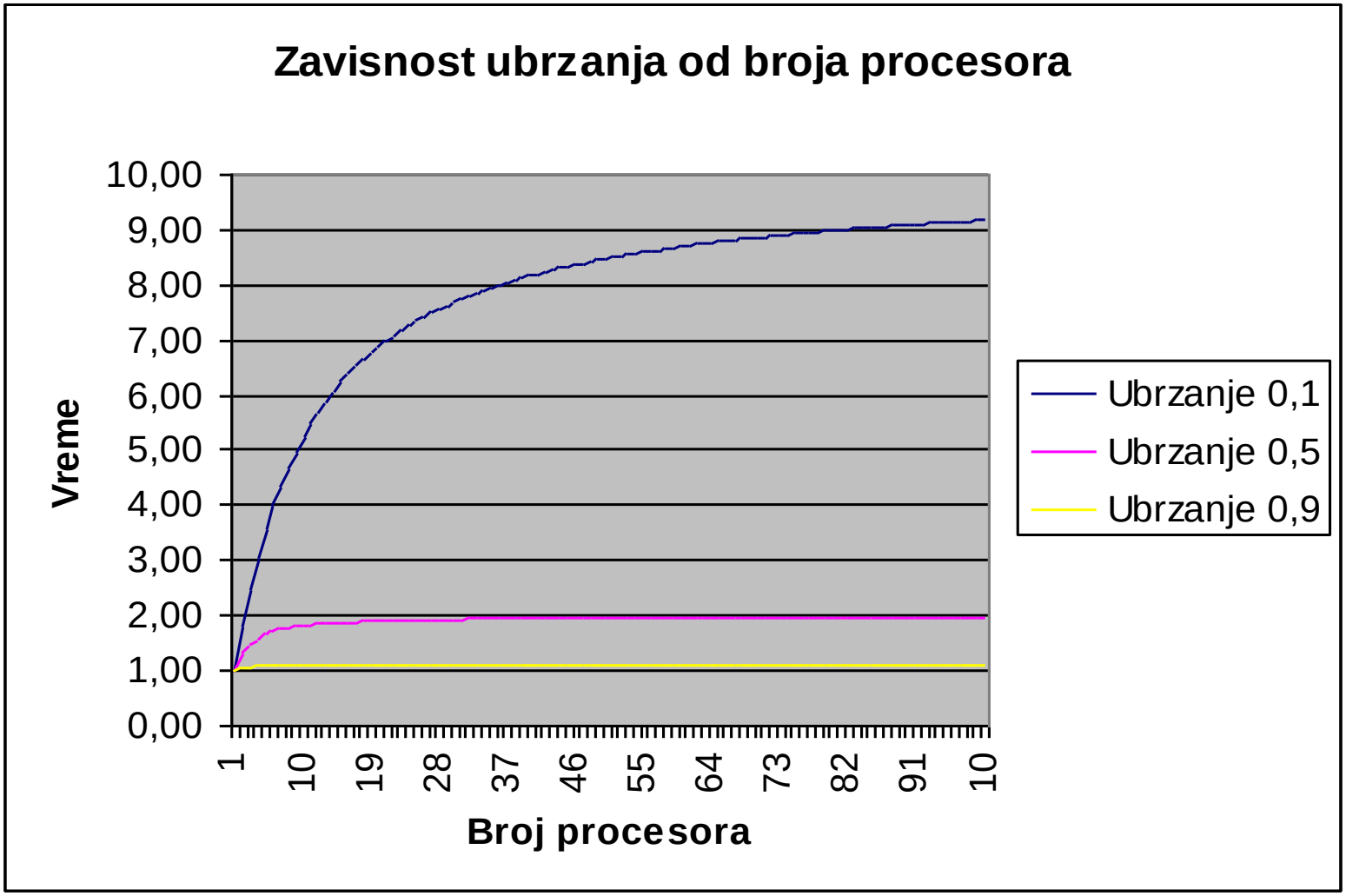
$$\lim_{p\to\infty} S_p = \lim_{p\to\infty} \frac{p}{1+(p-1)\cdot\alpha} = \lim_{p\to\infty} \frac{1}{\frac{1}{p}+\alpha-\frac{\alpha}{p}} = \frac{1}{\alpha}$$

- Efikasnost korišćenja procesora je S_p/p
- Ubrzanje je ograničeno sekvencijalnim delom aplikacije nezavisno od broja procesora

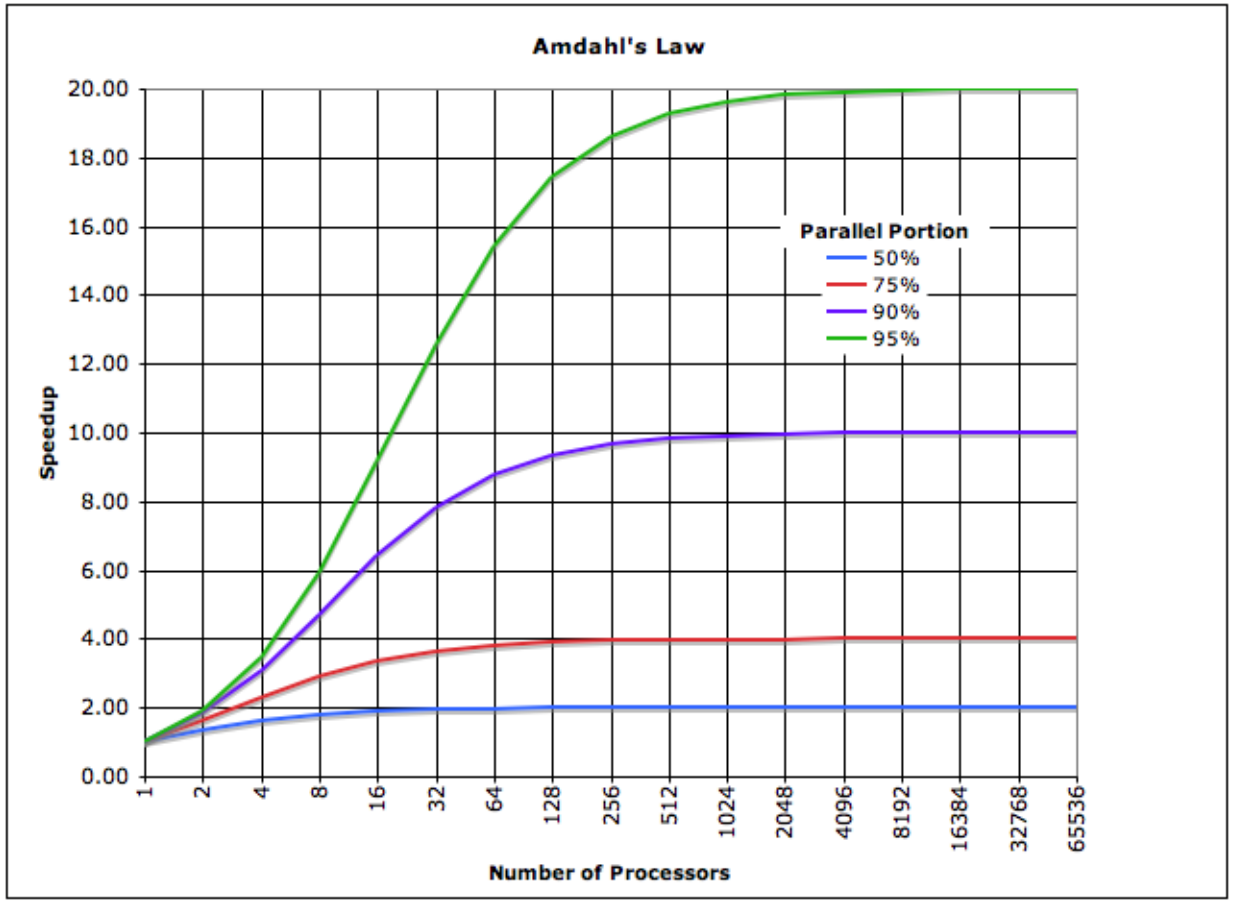
Vreme u zavisnosti od broja procesora i udela sekvencijalnog dela aplikacije



Ubrzanje u zavisnosti od broja procesora i udela sekvencijalnog dela aplikacije



Amdahl-ov zakon (3)



Amdahl-ov zakon (4)

- Ovaj zakon daje gornju granicu ubrzanja
- Ubrzanje ograničavaju:
  - Cena kreiranja paralelnih delova
  - Cena komunikacije deljenih podataka
  - Cena sinhronizacije
  - Cena redundantnih izračunavanja
  - Nebalansirano opterećenje (nedovoljan paralelizam, nejednake veličine paralelnih delova, ...)
- Potrebna krupna granularnost paralelizma
  - Ali ne previše!
- Ako je max ubrzanje S_p, obično S_p procesora radi sa 50% efikasnosti (empirijski)
- Da li je moguće superlinearno ubrzanje $S_p > p$?
  - Mnogo više operativne i keš memorije
  - Restrukturisanje algoritma pogodna za paralelizaciju

Skalabilnost

- Koliko efikasno aplikacija može da iskoristi povećanje nivoa paralelizma u sistemu?
  - Čvrsto skaliranje (*strong scaling*)
  - Labavo skaliranje (*weak scaling*)
- Čvrsto (jako) skaliranje
  - Ukupna veličina problema fiksna dok broj PE raste
  - Tipično za računski intenzivne programe (*compute bound*)
  - Pretpostavka *Amdahl-ovog* zakona
  - Ne odgovara uvek načinu korišćenja paralelnih računara
- Labavo (slabo) skaliranje
  - Veličina problema po jednom PE fiksna, ukupna raste
  - Tipično za memorijski intenzivne programe (*memory bound*)
  - Pretpostavka *Gustafson-Barsis*-ovog zakona

Gustafson-Barsis-ov zakon (1)

- Pretpostavka da paralelni deo aplikacije raste sa brojem procesora
  - Numeričke i grafičke aplikacije, prognoza vremena, itd.

$$T_p = \alpha T + (1 - \alpha)T$$

$$T_1 = \alpha T + (1 - \alpha)pT$$

$$S_p = \frac{T_1}{T_p} = \alpha + (1 - \alpha)p$$

$$S_p = p - (p - 1)\alpha$$

- Teoretsko usporenje već paralelizovanog zadataka
  - Kada bi se izvršavao na sekvencijalnoj mašini

Gustafson-Barsis-ov zakon (2)

- Pretpostavlja da se izračunavanja koja koriste dovoljno velike skupove podataka mogu efikasno paralelizovati
  - Kontrast u odnosu na Amdalov zakon
  - Programeri imaju tendenciju povećanja veličine problema sa rastom računarskih resursa
  - Treba odrediti veličinu problema spram karakteristika dostupnog paralelnog hardvera
    - Kako bi se rešili veći problemi u fiksnom vremenu
- Implikacije
  - Odnosi se samo na skalabilne aplikacije
  - I dalje treba smanjivati sekvencijalni deo!

Trendovi razvoja paralelnih sistema

Naučne aplikacije

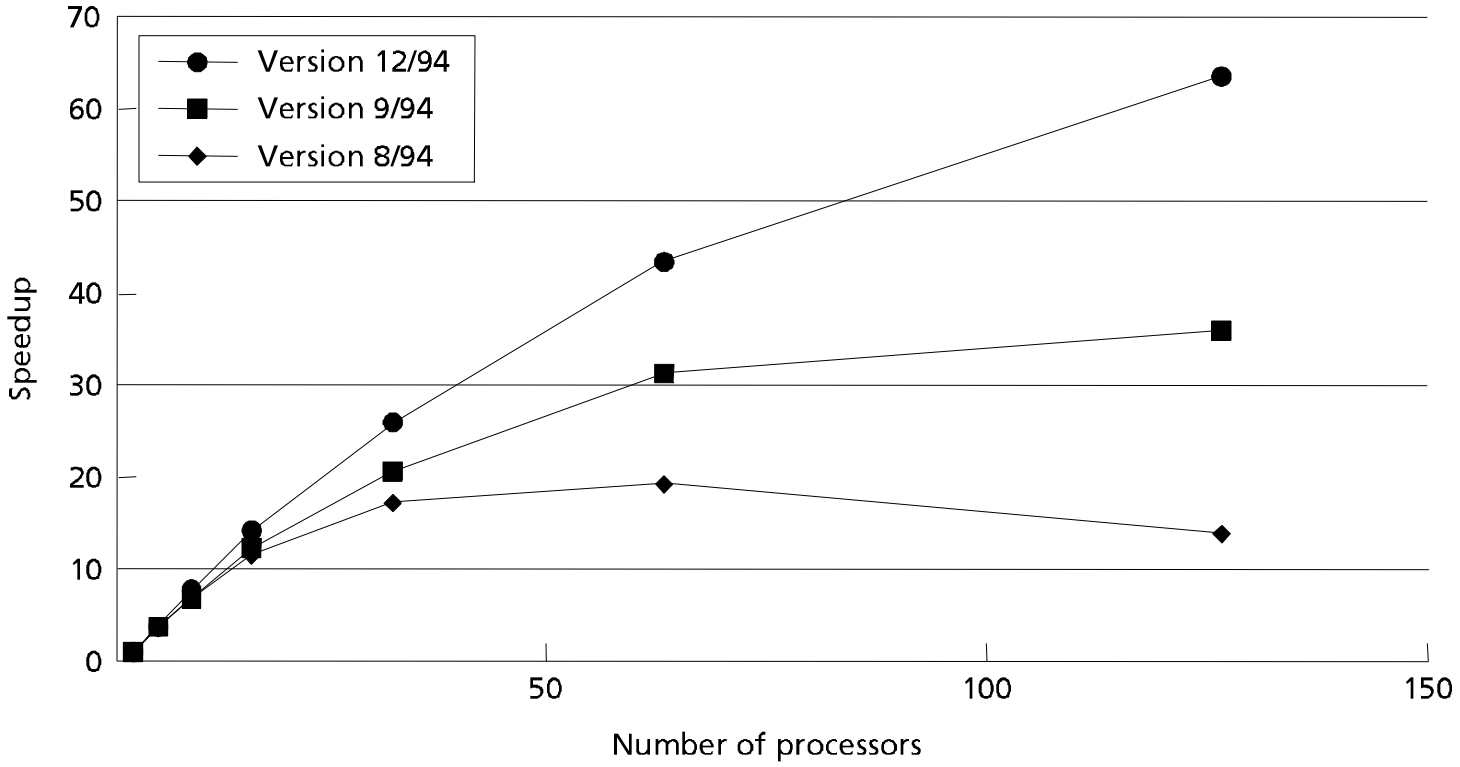
- Zahtevaju ogromnu procesorsku snagu i ogroman memorijski prostor:
  - Nuklearna fizika
  - Kvantna dinamika
  - Dinamika fluida
  - Klimatski modeli i vremenska prognoza
  - Okeanske struje
  - Seizmologija
  - Molekularna biologija
  - Analiza genoma
  - Superprovodnici
  - Robotika

Inženjerske aplikacije

- Veliki paralelni računari neophodni u mnogim industrijama:
  - Naftna (analiza rezervoara)
  - Automobilska (simulacija sudara, …)
  - Aeronautička (aerodinamički modeli, strukturalna mehanika)
  - *Computer Aided Design* (CAD) softver
  - Farmaceutska (modeliranje molekula)
  - Modelovanje ekonomskih procesa
  - Bankarstvo, osiguranje, analiza rizika
  - Regresivni testovi za veliki SW
  - Vizuelizacija
    - U svim pomenutim slučajevima
    - Industrija zabave
    - Arhitektura, dizajn enterijera i industrijski dizajn

Interakcija aplikacija-arhitektura

- AMBER - program za simulaciju dinamike molekula
- Početna tačka - vektorski kod za Cray-1
- 145 MFLOPs - Cray90, 406 MFLOPs - 128-processor Paragon, 891 MFLOPs - 128-processor Cray T3D



Komercijalne aplikacije

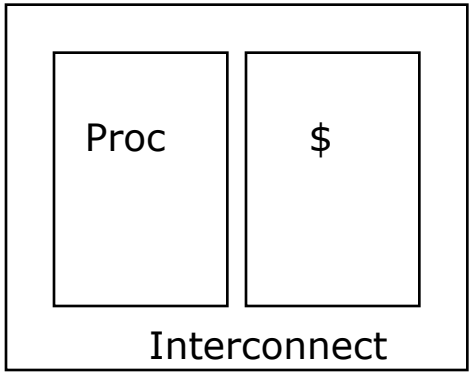
- Zasnovane na paralelizmu sve većeg obima
  - Procesna snaga određuje obim posla koji može da se uradi
  - Serveri baza podataka
  - OLTP obrada transakcija
  - Podrška za sisteme odlučivanja
  - "*data mining*"
  - "*data warehousing*" ...
- Aplikacije za lične prenosne uređaje
  - Multimedija (govor, zvuk, slika, video!)
  - Zabavne
  - Zdravstvene
  - Poslovne
  - Društvene mreže

Trendovi aplikacije - rezime

- Prelazak na paralelno procesiranje za naučne i inženjerske aplikacije
- Vrlo zastupljeno i u komercijalnim aplikacijama
  - baze podataka, transakcije, finansije
  - manji, ali i veći sistemi
- Mobilne platforme uveliko izvršavaju višenitne (*multithreaded*) programe, koji vrlo liče na paralelne programe
- Zahtevi za efikasnije izvršavanje sekvencijalnih aplikacija
  - Mali MP, CMPs, TLS
- Zahtevi aplikacija su veliki i biće sve veći

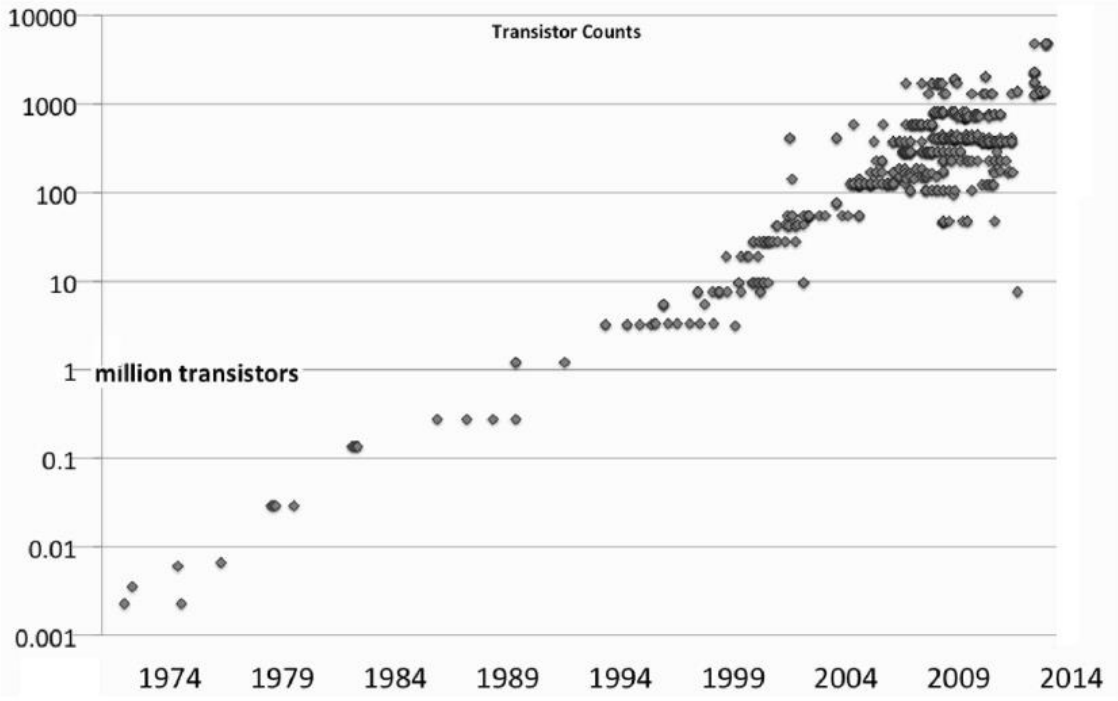
Trendovi tehnologije

- *Feature size* (λ)
  - Minimalna veličina tranzistora ili žice po x ili y osi
  - Osnovni napredak – smanjenje λ
    10 μm (1971) -> 32 nm (2011) -> 3nm (2024)
  - Performansa tranzistora skalira linearno
    - Kašnjenje po žici se ne popravlja sa smanjenjem λ!
  - Gustina integracije skalira sa λ^2 (čak i brže)
- Performanse > 100x po dekadi
  - takt < 10x, ostalo doprinos broja tranzistora
- Kako iskoristiti sve više tranzistora?
  - Paralelna obrada
    - Više operacija po ciklusu smanjuje CPI
  - Lokalnost u pristupu podacima
    - Izbegava latenciju i smanjuje CPI
    - Poboljšava iskorišćenje procesora
  - Za oboje potrebni resursi - kompromis

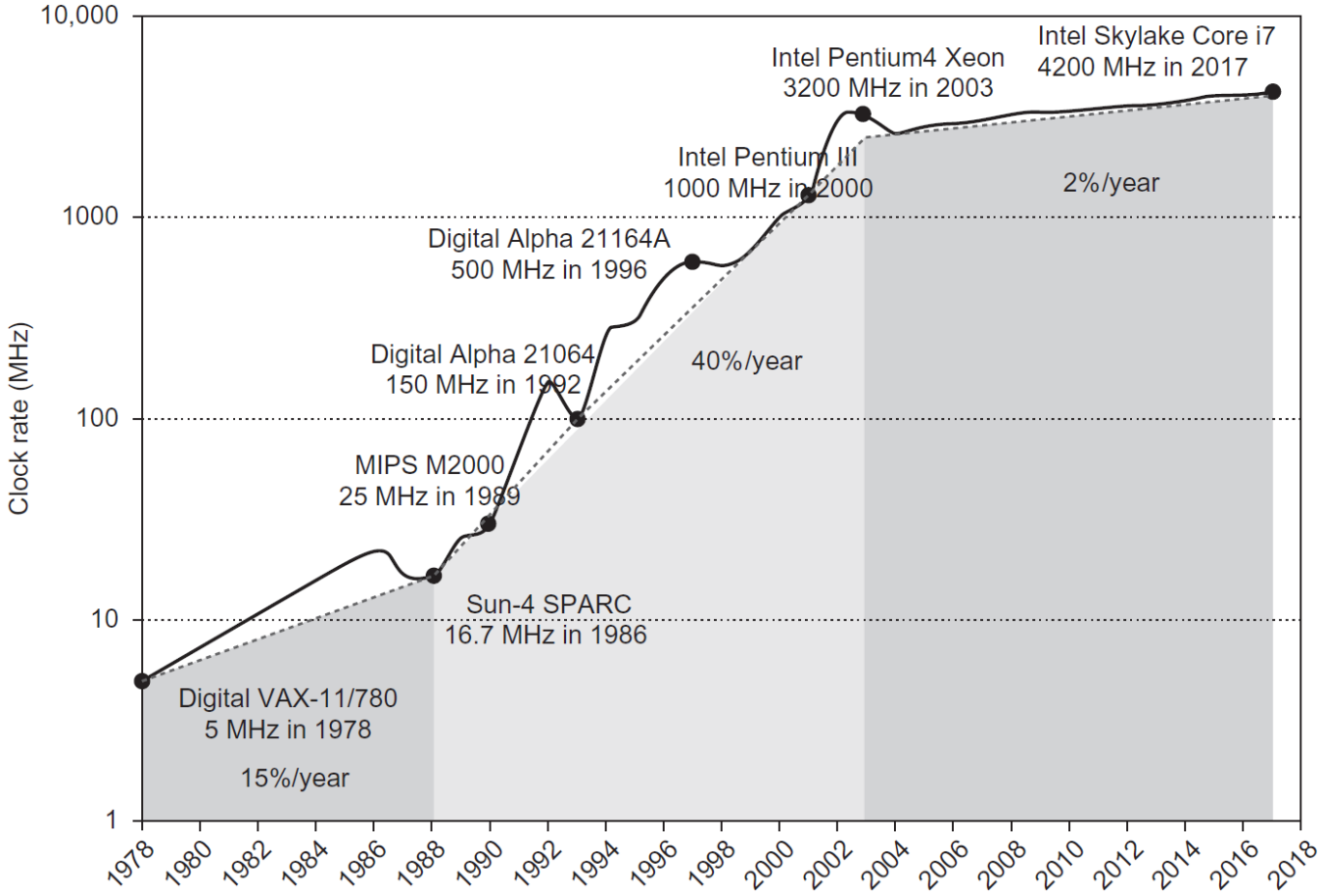


Trendovi tehnologije

- Tehnologija integrisanih kola
  - Gustina tranzistora: 35%/g.
  - Veličina čipa: 10-20%/g.
  - Ukupno: 40-55%/g. (x2/1.5-2g. *Moore-ov* zakon)



Trendovi tehnologije

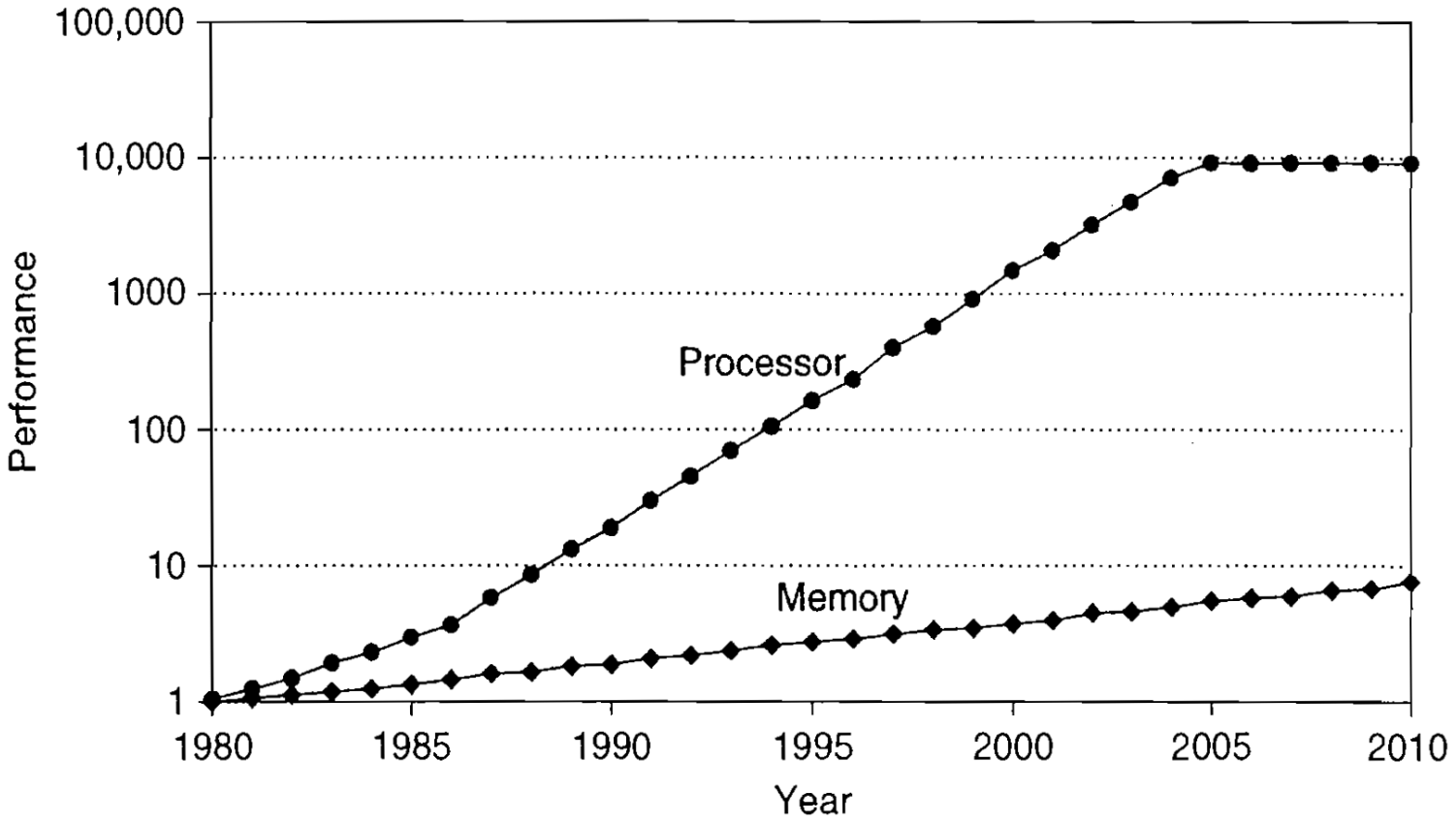


- Zasićenje ubrzavanja takta (usled *power wall*)

Trendovi u memorijskoj tehnologiji

- Disparitet između kapaciteta i brzine memorije sve izraženiji
  - Kapacitet porastao 1000x 1980-95, brzina samo 2x (oko 7% godišnje)
  - Razlika između brzine CPU i memorije sve veća
- Paralelizam i lokalnost u memorijskim sistemima
  - Treba preneti više podataka u paraleli brzim "pipeline" prenosom kroz relativno "uski" interfejs
  - Dublje hijerarhije keš memorija
  - Organizacija keš memorija?
  - Baferske keš memorije sa najskorije korišćenim podacima
- Paralelizam povećava efektivnu veličinu svakog nivoa hijerarhije, bez produženja latencije
- Fleš memorije
  - Kapaciteti rastu 50-60% godišnje (15-20X jeftiniji po bitu od DRAM)
- Hard diskovi – paralelizam (npr. RAID)+ keširanje
  - Kapaciteti rastu do 40 % godišnje (300-500X jeftiniji po bitu od DRAM)

Trendovi u memorijskoj tehnologiji



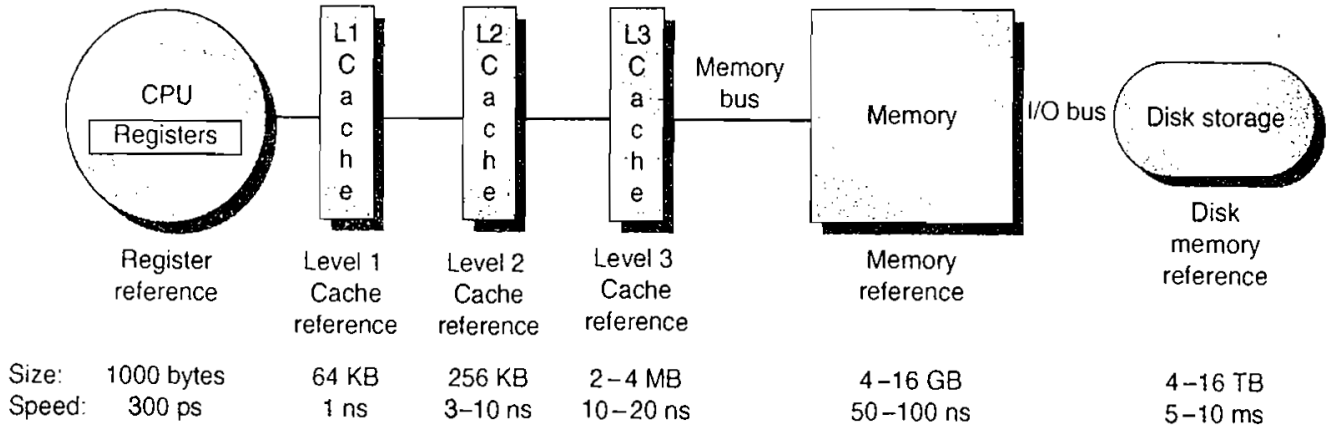
- CPU – memory gap

Trendovi u memorijskoj tehnologiji

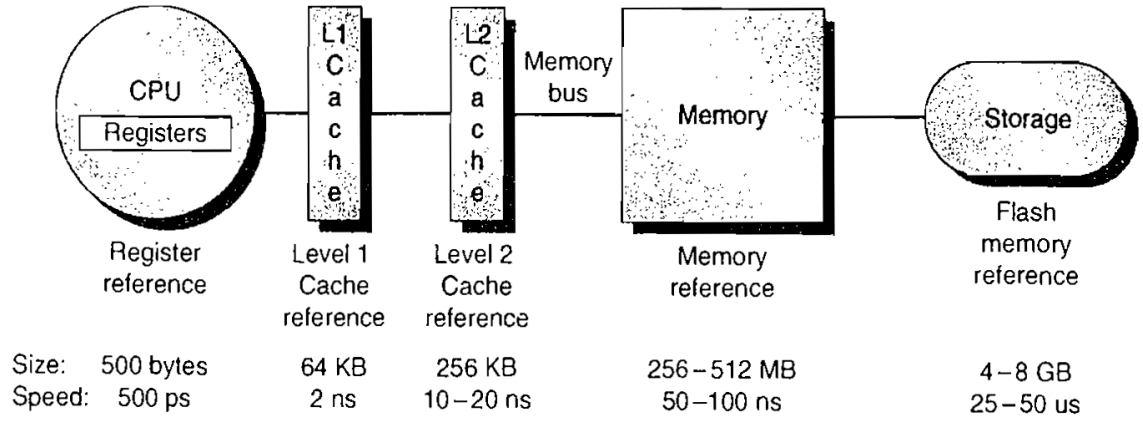
- Usporavanje rasta kapaciteta DRAM memorije
  - Kapacitet porastao 1000x 1980-95, brzina samo 2x
  - Gigabit DRAM, ali razlika između brzine CPU i memorije sve veća

Year	DRAM growth rate	Characterization of impact on DRAM capacity
1990	60%/year	Quadrupling every 3 years
1996	60%/year	Quadrupling every 3 years
2003	40%–60%/year	Quadrupling every 3 to 4 years
2007	40%/year	Doubling every 2 years
2011	25%–40%/year	Doubling every 2 to 3 years

Trendovi u memorijskoj tehnologiji



(a) Memory hierarchy for server



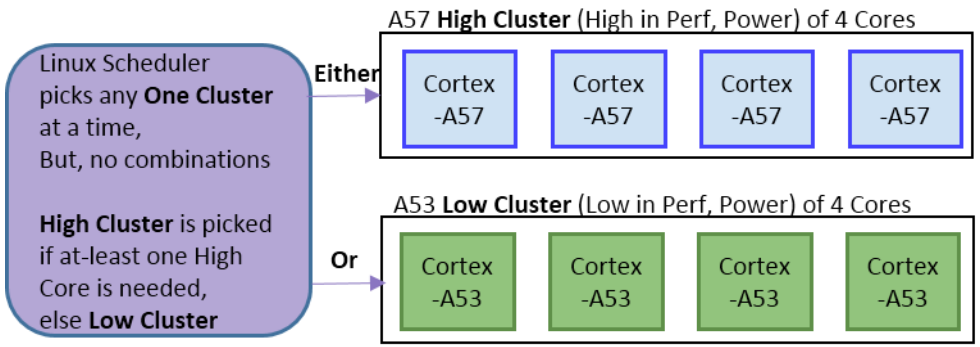
(b) Memory hierarchy for a personal mobile device

Trendovi tehnologije

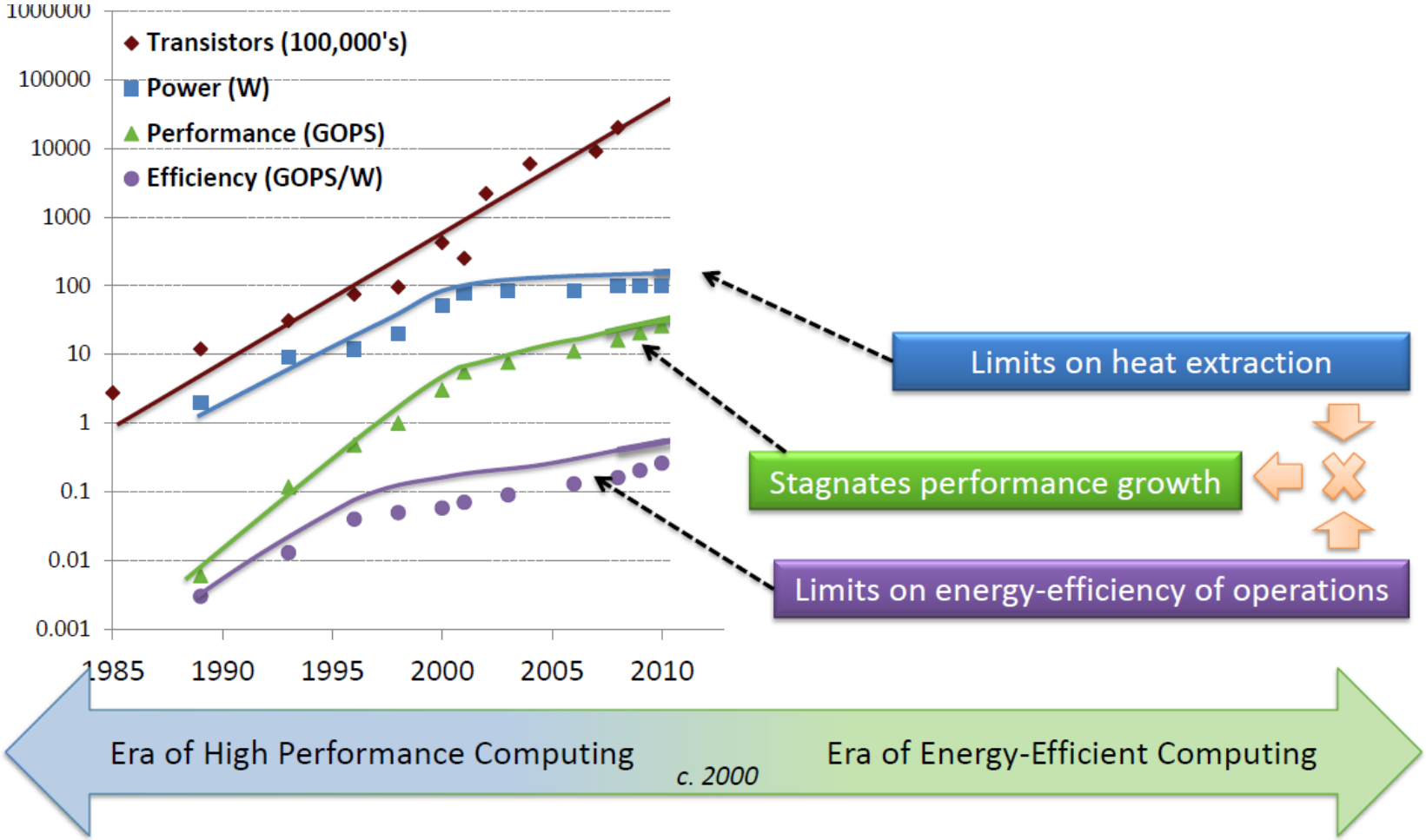
- Problemi: dovesti energiju, odvesti disipaciju
- Potrošnja
  - Procesor u mobilnom telefonu 0.5-2W
  - Intel 80386 ~ 2 W
  - 3.3 GHz Intel Core i7 do 130 W
  - Superračunari ~ MW
- Toplota mora biti disipirana sa 1.5 x 1.5 cm čipa
- Ovo je granica za vazdušno hlađenje
- Statička snaga
  - Srazmerna naponu i broju tranzistora
  - Struje curenja
- Dinamička snaga
  - Srazmerna kvadratu napona (za 20g. sa 5V na 1V)
  - Srazmerna frekvenciji

Trendovi tehnologije

- Tehnike za smanjivanje snage
  - Isključivanje takta za neaktivne module
  - Režimi rada sa smanjenom frekvencijom
    - *Dynamic Voltage-Frequency Scaling* pogotovo kod PMD
  - Projektovanje za tipičan slučaj sa sigurnosnim mehanizmom
    - *emergency slowdown*
  - Privremeni *turbo* režim kada je to sigurno
    - npr. *overclocking* sa 3.3GHz na 3.6GHz kod i7
- Asimetrične arhitekture
  - *Low-power* vs. *performance cores*
  - ARM big.LITTLE arhitektura
    - *Cortex-A7, A53m A57*
  - ARM *DynamIQ*
    - *Cortex-A75, A55*
  - Intel *Alder Lake*



Trendovi tehnologije



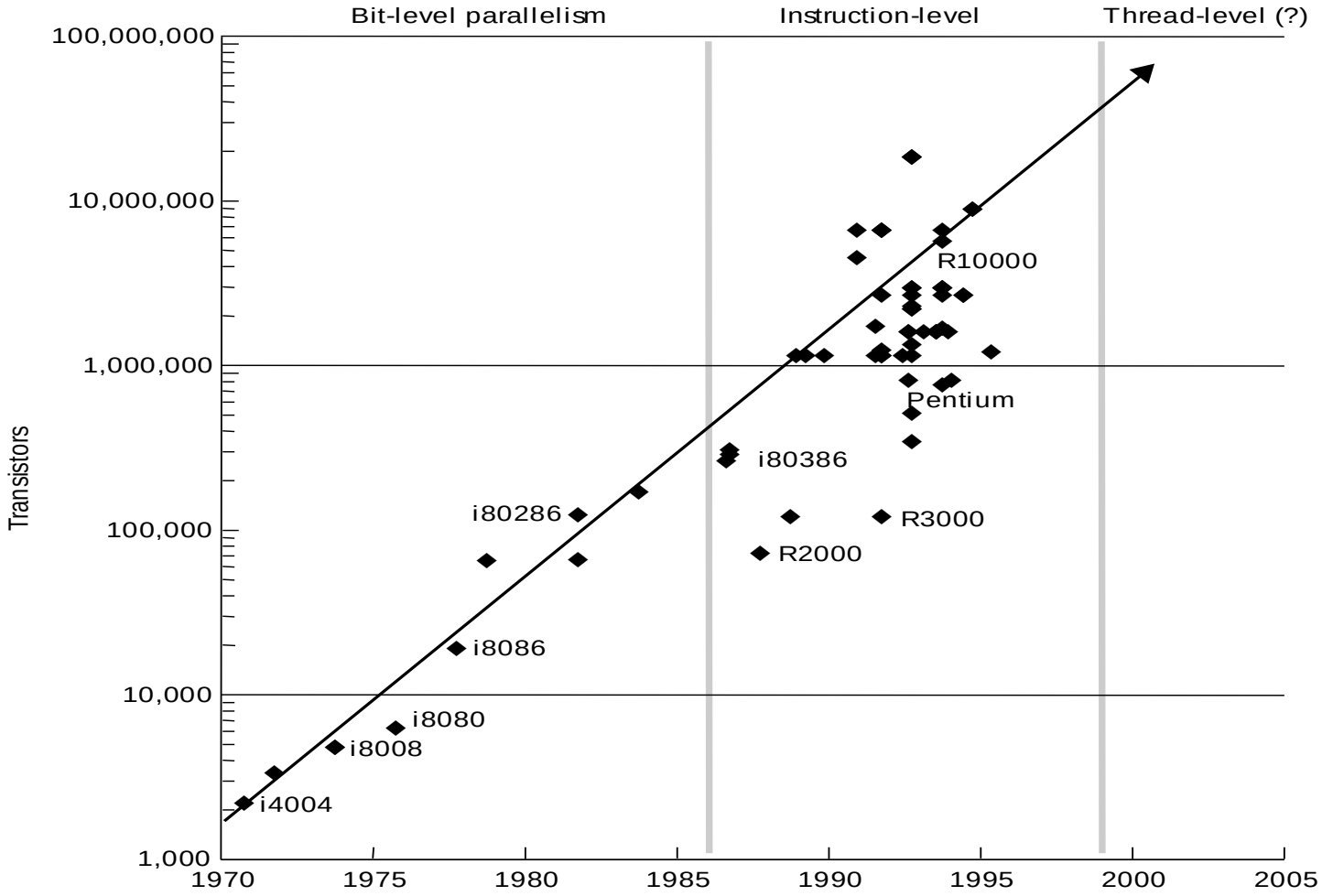
Trendovi tehnologije

- Ograničenja Si tehnologije (1 ns na 3cm)
- Energetska efikasnost i disipacija (*power wall*)
- Disparitet brzine CPU i memorije (*memory wall*)
- Sa smanjenjem λ, pitanje interne pouzdanosti
- Troškovi projektovanja i verifikacije
- Neminovnost CMP (*multicore*, *manycore*)
- 1000+ procesora na čipu moguće
- Cisco mrežni procesor
  - 188 procesora, 5-stage Tensilica, 1/3 DRAM i spec. FU
  - 130nm, 18x18mm, 35W, 250MHz, 50GIPS

Trendovi arhitekture

- Arhitektura prevodi "darove" tehnologije u performanse i mogućnosti
- Razrešava kompromis (*trade-off*) između paralelizma i lokalnosti
  - Optimum obično po sredini
  - Mikroprocesor: 1/3 obrada, 1/3 keš, 1/3 *off-chip* sprega
  - Kompromis varira zavisno od obima sistema i napretka tehnologije
- Razumevanje trendova arhitekture mikroprocesora
  - Daje intuiciju za projektne odluke u paralelnim sistemima
  - Pokazuje osnovnu ulogu paralelizma i u "sekvencijalnim" računarima

VLSI – IV generacija

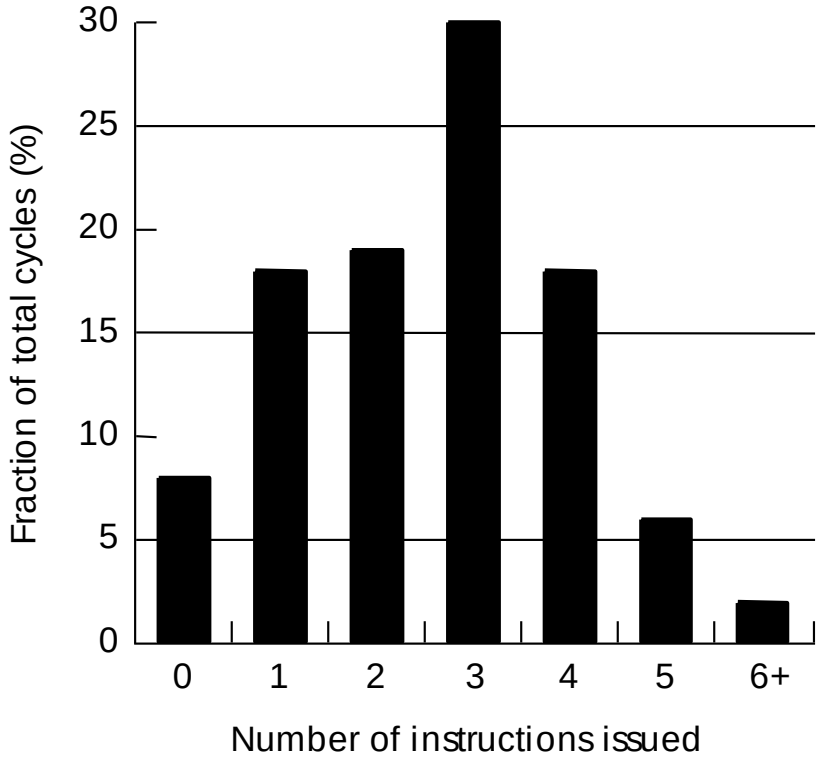


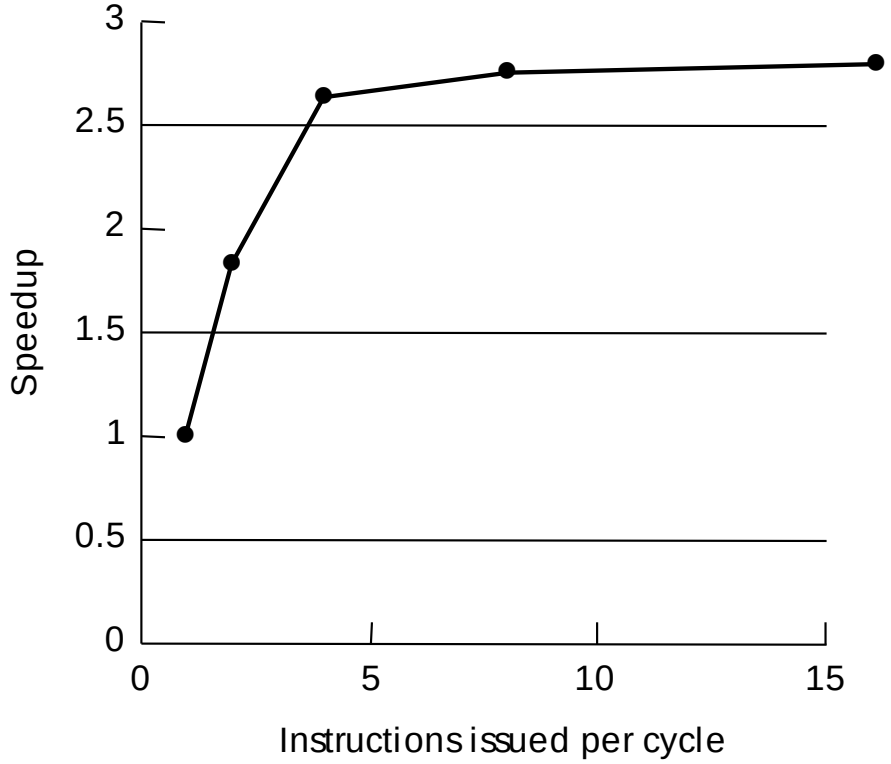
Trendovi arhitekture

- Prepoznatljiv trend u VLSI je povećavanje nivoa paralelizma
  - ... 1985: bit-paralelizam (širina staze podataka):
    - 4-bit >> 8-bit >> 16-bit >> 32-bit > usporava ka 64
    - smanjuje se broj ciklusa u obradi
    - potreba za 128-bit nije urgentna (1-addr.bit godišnje)
    - ključna tačka - kad pun 32-bit micro i keš staju na čip (’86)
  - 80-te ...90 -te: ILP
    - protočna obrada i “proste” instrukcije + napredak prevodioca (RISC)
    - *on-chip* keš i više FU => superskalar
    - veliko usložnjenje: *out-of-order* izvršavanje, spekulacija, predikcija
      - problemi – kontrola toka i latencija
      - veliki zahevi za resurse na čipu
  - Zatim >> paralelizam na nivou niti (*thread*) - TLP

Potencijali ILP-a

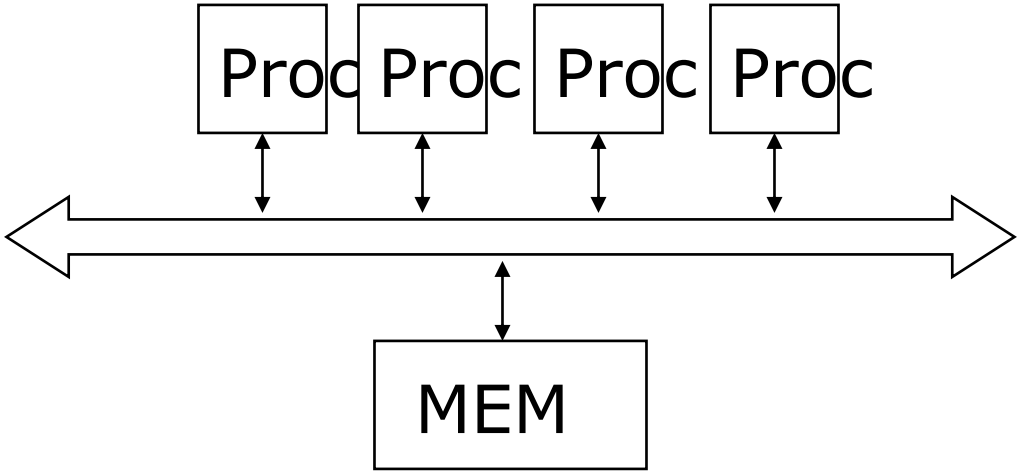
- Simulacije na "idealnoj" mašini bez ograničenja na resurse (FU) i prihvatanje instrukcija, "*renaming*", 100% predikcija skoka – bez zastoja izvršavanja (J'91)





TLP "on board"

- Mikroprocesor prirodan gradivni element MP sistema sa zajedničkom memorijom
  - Olakšano povezivanje (npr. Pentium-Pro)
- Brži procesori zasićuju magistralu, zato unapređenje ove tehnologije 90-tih
  - Dominira tržištem servera i poslovanih sistema, ide ka desktopu
  - Širok opseg sistema od CMP do velikog obima

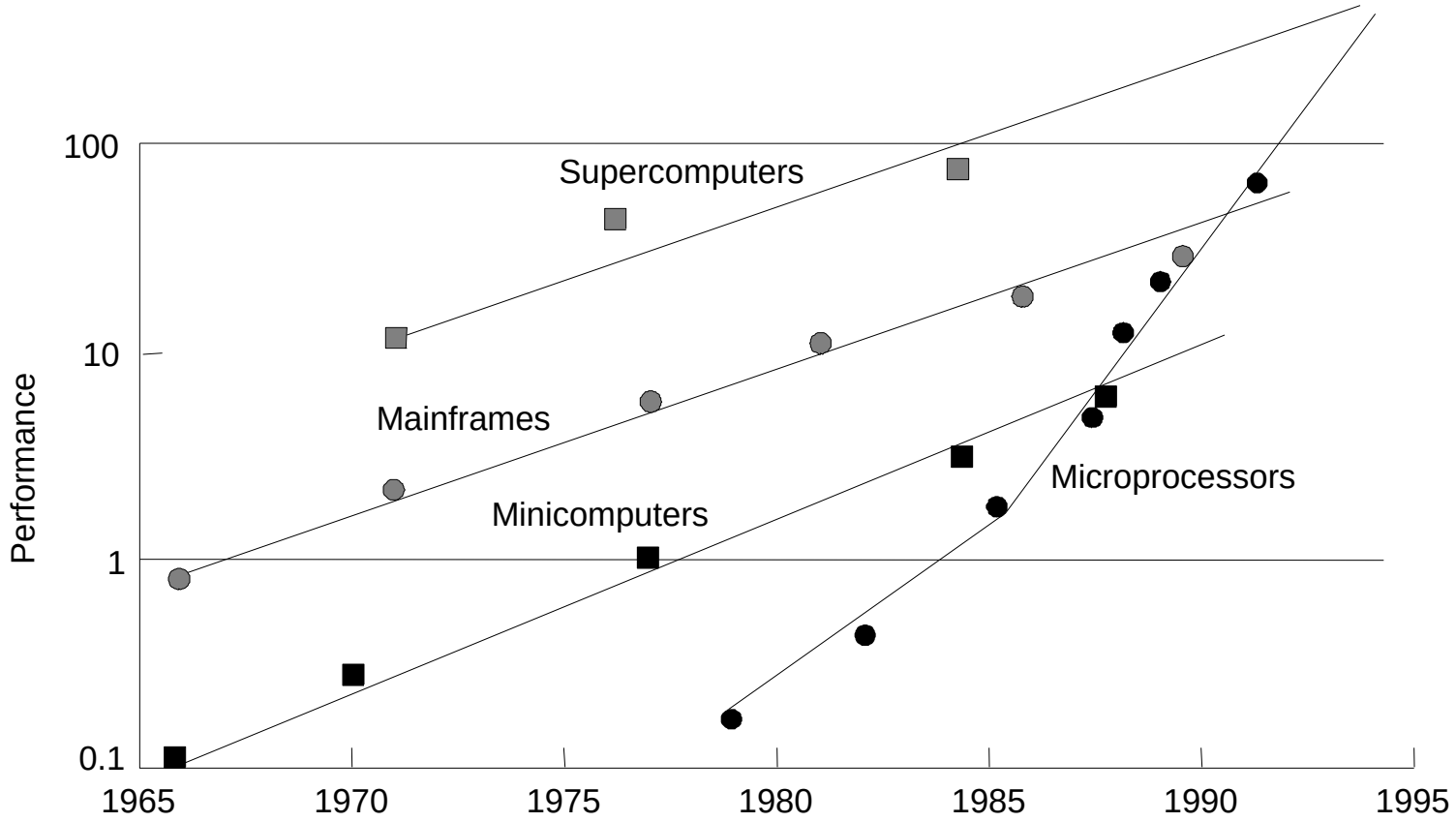


Ekonomičnost

- Mikroprocesori na tržištu ne samo brzi i moćni, nego i jeftini
  - Cene razvoja – 10x M$
  - Dupliranje serije - 10% manja cena primerka
  - Masovno se prodaju za razliku od superračunara
  - Iskoristiti ulaganje i upotrebiti kao gradivni blok
- Multiprocesori "pogurani" od SW tržišta
  - (npr., serveri BP) kao i od HW tržišta
- Mali SMP sa zajedničkom magistralom postao standardan artikal
- Multiprocesor na čipu je odavno realnost!

Ekonomičnost

- Mikroprocesor kao gradivni blok za MP!



Trendovi performansi

Microprocessor	16-Bit address/ bus, microcoded	32-Bit address/ bus, microcoded	5-Stage pipeline, on-chip I & D caches, FPU	2-Way superscalar, 64-bit bus	Out-of-order 3-way superscalar	Out-of-order superpipelined, on-chip L2 cache	Multicore OOO 4-way on chip L3 cache, Turbo
Product	Intel 80286	Intel 80386	Intel 80486	Intel Pentium	Intel Pentium Pro	Intel Pentium 4	Intel Core i7
Year	1982	1985	1989	1993	1997	2001	2015
Die size (mm^2)	47	43	81	90	308	217	122
Transistors	134,000	275,000	1,200,000	3,100,000	5,500,000	42,000,000	1,750,000,000
Processors/chip	1	1	1	1	1	1	4
Pins	68	132	168	273	387	423	1400
Latency (clocks)	6	5	5	5	10	22	14
Bus width (bits)	16	32	32	64	64	64	196
Clock rate (MHz)	12.5	16	25	66	200	1500	4000
Bandwidth (MIPS)	2	6	25	132	600	4500	64,000
Latency (ns)	320	313	200	76	50	15	4
Memory module	DRAM	Page mode DRAM	Fast page mode DRAM	Fast page mode DRAM	Synchronous DRAM	Double data rate SDRAM	DDR4 SDRAM
Module width (bits)	16	16	32	64	64	64	64
Year	1980	1983	1986	1993	1997	2000	2016
Mbits/DRAM chip	0.06	0.25	1	16	64	256	4096
Die size (mm^2)	35	45	70	130	170	204	50
Pins/DRAM chip	16	16	18	20	54	66	134
Bandwidth (MBytes/s)	13	40	160	267	640	1600	27,000
Latency (ns)	225	170	125	75	62	52	30

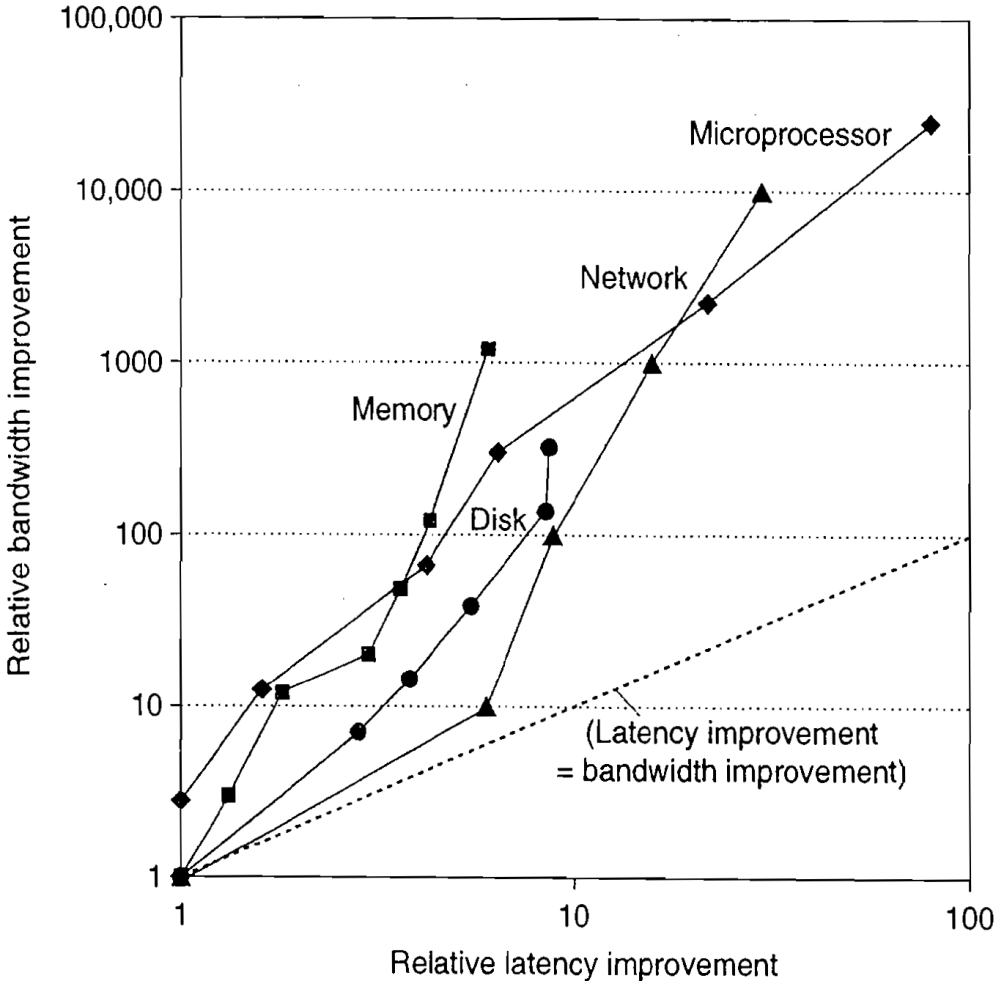
Trendovi performansi

Local area network	Ethernet	Fast Ethernet	Gigabit Ethernet	10 Gigabit Ethernet	100 Gigabit Ethernet	400 Gigabit Ethernet
IEEE standard	802.3	803.3u	802.3ab	802.3ac	802.3ba	802.3bs
Year	1978	1995	1999	2003	2010	2017
Bandwidth (Mbits/seconds)	10	100	1000	10,000	100,000	400,000
Latency (μs)	3000	500	340	190	100	60
Hard disk	3600 RPM	5400 RPM	7200 RPM	10,000 RPM	15,000 RPM	15,000 RPM
Product	CDC WrenI 94145-36	Seagate ST41600	Seagate ST15150	Seagate ST39102	Seagate ST373453	Seagate ST600MX0062
Year	1983	1990	1994	1998	2003	2016
Capacity (GB)	0.03	1.4	4.3	9.1	73.4	600
Disk form factor	5.25 in.	5.25 in.	3.5 in.	3.5 in.	3.5 in.	3.5 in.
Media diameter	5.25 in.	5.25 in.	3.5 in.	3.0 in.	2.5 in.	2.5 in.
Interface	ST-412	SCSI	SCSI	SCSI	SCSI	SAS
Bandwidth (MBytes/s)	0.6	4	9	24	86	250
Latency (ms)	48.3	17.1	12.7	8.8	5.7	3.6

Trendovi performansi

- *Bandwidth* ili *throughput*
  - Ukupan posao urađen za dato vreme
  - 10,000-25,000x poboljšanje za procesore
  - 300-1200x poboljšanje za memorije i diskove
- Latencija ili vreme odgovora
  - Vreme od početka do završetka događaja
  - 30-80x poboljšanje kod procesora
  - 6-8x poboljšanje za memorije i diskove
- *Bandwidth* se brže poboljšava od latencije
  - Otprilike kvadratno

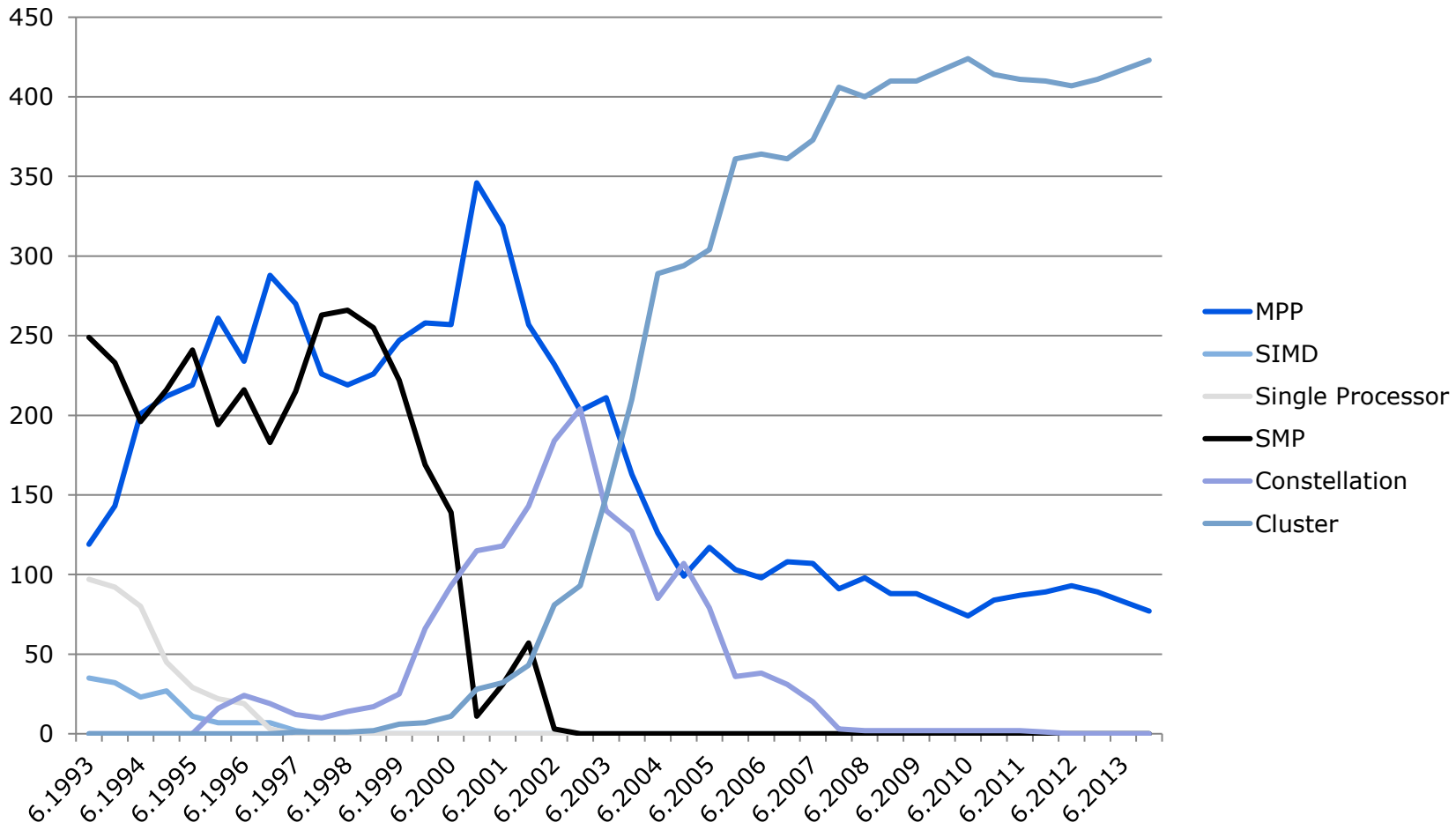
Performanse (1)



Performanse (2)

- Superračunari - zahtev za maksimalnim performansama
- Istorijski uvodili i verifikovali inovativne arhitekture i tehnike (protočna obrada, itd.)
  - Manje tržište od komercijalnih
  - Dominiraju vektorski računari počevši od 70-tih
  - Microprocesori ostvarili veliki napredak u FP performansama zahvaljujući:
    - Ubrzanju takta
    - Protočnim FPU
    - ILP
    - Sve efektivnijem keširanju
  - Plus ekonomičnost
- MPP zamenjuju vektorske superračunare

Top500 lista



Top500 lista (June 2024)

- Lista najmoćnijih superračunarskih sistema
  - Prema *Linpack benchmark* testu

Rank	System	Cores	Rmax (PFlop/s)	Rpeak (PFlop/s)	Power (kW)
1	**Frontier** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, **HPE** DOE/SC/Oak [Top500 lista] tional Laboratory **United States**	8,699,904	1,206.00	1,714.81	22,786
2	**Aurora** - HPE Cray EX - Intel Exascale Compute Blade, Xeon CPU Max 9470 52C 2.4GHz, Intel Data Center GPU Max, Slingshot-11, **Intel** DOE/SC/Argonne National Laboratory **United States**	9,264,128	1,012.00	1,980.01	38,698
3	**Eagle** - Microsoft NDv5, Xeon Platinum 8480C 48C 2GHz, NVIDIA H100, NVIDIA Infiniband NDR, **Microsoft Azure** Microsoft Azure **United States**	2,073,600	561.20	846.84	
4	**Supercomputer Fugaku** - Supercomputer Fugaku, A64FX 48C 2.2GHz, Tofu interconnect D, **Fujitsu** RIKEN Center for Computational Science **Japan**	7,630,848	442.01	537.21	29,899
5	**LUMI** - HPE Cray EX235a, AMD Optimized 3rd Generation EPYC 64C 2GHz, AMD Instinct MI250X, Slingshot-11, **HPE** EuroHPC/CSC **Finland**	2,752,704	379.70	531.51	7,107

Top500 - Frontier - HPE Cray EX235a, AMD (June 2024)

Site:	DOE/SC/Oak Ridge National Laboratory
System URL:	https://www.olcf.ornl.gov/frontier/
Manufacturer:	HPE
Cores:	8,699,904
Processor:	AMD Optimized 3rd Generation EPYC 64C 2GHz
Interconnect:	Slingshot-11
Installation Year:	2021
Performance	
Linpack Performance (Rmax)	1,206.00 PFlop/s
Theoretical Peak (Rpeak)	1,714.81 PFlop/s
Nmax	24,330,240
HPCG	14,054.0 TFlop/s
Power Consumption	
Power:	22,786.00 kW

Top500 - Summit - IBM Power System AC922 (November 2019)

Site:	DOE/SC/Oak Ridge National Laboratory
System URL:	http://www.olcf.ornl.gov/olcf-resources/compute-systems/summit/
Manufacturer:	IBM
Cores:	2,414,592
Memory:	2,801,664 GB
Processor:	IBM POWER9 22C 3.07GHz
Interconnect:	Dual-rail Mellanox EDR Infiniband
Performance	
Linpack Performance (Rmax)	148,600 TFlop/s
Theoretical Peak (Rpeak)	200,795 TFlop/s
Nmax	16,473,600
HPCG [TFlop/s]	2,925.75
Power Consumption	
Power:	10,096.00 kW (Submitted)
Power Measurement Level:	3
Measured Cores:	2,397,824

Green500 – potrošnja energije (June 2024)

Rank	TOP500 Rank	System	Cores	Rmax (PFlop/s)	Power (kW)	Energy Efficiency (GFlops/watts)
1	189	**JEDI** - BullSequana XH3000, Grace Hopper Superchip 72C 3GHz, NVIDIA GH200 Superchip, Quad-Rail NVIDIA InfiniBand NDR200, ParTec/EVIDEN EuroHPC/FZJ Germany	19,584	4.50	67	72.733
2	128	**Isambard-AI phase 1** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE University of Bristol United Kingdom	34,272	7.42	117	68.835
3	55	**Helios GPU** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Cyfronet Poland	89,760	19.14	317	66.948
4	328	**Henri** - ThinkSystem SR670 V2, Intel Xeon Platinum 8362 32C 2.8GHz, NVIDIA H100 80GB PCIe, Infiniband HDR, Lenovo Flatiron Institute United States	8,288	2.88	44	65.396
5	71	**preAlps** - HPE Cray EX254n, NVIDIA Grace 72C 3.1GHz, NVIDIA GH200 Superchip, Slingshot-11, HPE Swiss National Supercomputing Centre (CSCS) Switzerland	81,600	15.47	240	64.381

Razlozi za paralelne arhitekture

- Sve veća zastupljenost
  - Ekonomičnost, tehnologija, arhitektura, zahtevi aplikacije
- Sve platforme - MP
- Paralelizam se koristi na više nivoa:
  - ILP
  - Multiprocesorski serveri
  - MPP
- Perspektiva sa strane memorijskog sistema slična perspektivi sa strane procesora
  - Povećana propusna moć, smanjena prosečna latencija sa lokalnim memorijama
  - Cena komunikacije
- Spektar paralelnih arhitektura
  - Opseg cena, performanse i skalabilinost

Ograničenja paralelnog programiranja (1)

- Složenost
  - Paralelne aplikacije su
    za red veličine složenije od sekvencijalnih programa
  - Ne samo da postoji više tokova izvršavanja programa,
    već postoji i tok podataka među njima
  - Paralelno programiranje zahteva da
    programer potroši više vremena prilikom
    dizajna programa, kodiranja, debagovanja,
    podešavanja performansi i održavanja
  - Pridržavanje uobičajenih tehnika razvoja softvera je
    značajno ako na projektu radi veći tim

Ograničenja paralelnog programiranja (2)

- Prenosivost (portability)
  - Situacija je prilično dobra, jer postoji nekoliko standarda za paralelno programiranje
  - MPI, POSIX threads, HPF, OpenMP standardi
  - Problem sa različitim implementacijama standarda na različitim platformama
  - Značajan uticaj operativnih sistema
  - Hardverske platforme mogu biti značajno različite i mogu uticati na prenosivost

Ograničenja paralelnog programiranja (3)

- Skalabilnost
  - Prosto dodavanje novih resursa (procesora) često nije rešenje
    - Može dovesti do zasićenja ili opadanja performansi zbog niza faktora
  - Korišćeni algoritam po svojoj prirodi može imati ograničenja po pitanju skalabilnosti
- Hardver značajno utiče na skalabilnost
  - Propusni opseg magistrale na SMP mašinama
  - Propusni opseg komunikacione mreže
  - Ukupna raspoloživa memorija
  - Brzina (takt) procesora
- Paralelne biblioteke mogu ograničiti skalabilnost

Ograničenja paralelnog programiranja (4)

- Zahtevi za resursima
  - Da bi se smanjilo ukupno vreme izvršavanja programa potrebno je potrošiti više procesorskog vremena
  - Paralelni programi mogu zahtevati više memorije za svoje izvršavanje
    - Replikacija podataka na više procesora
    - Dodatni trošak za paralelne biblioteke i podsisteme
- Režijski troškovi paralelizacije (*overhead*)
  - Kreiranje zadataka, komunikacija, pokretanje paralelnog okruženja
  - Paralelizacija često nije rešenje za probleme malih dimenzija!

Primeri paralelnih programa

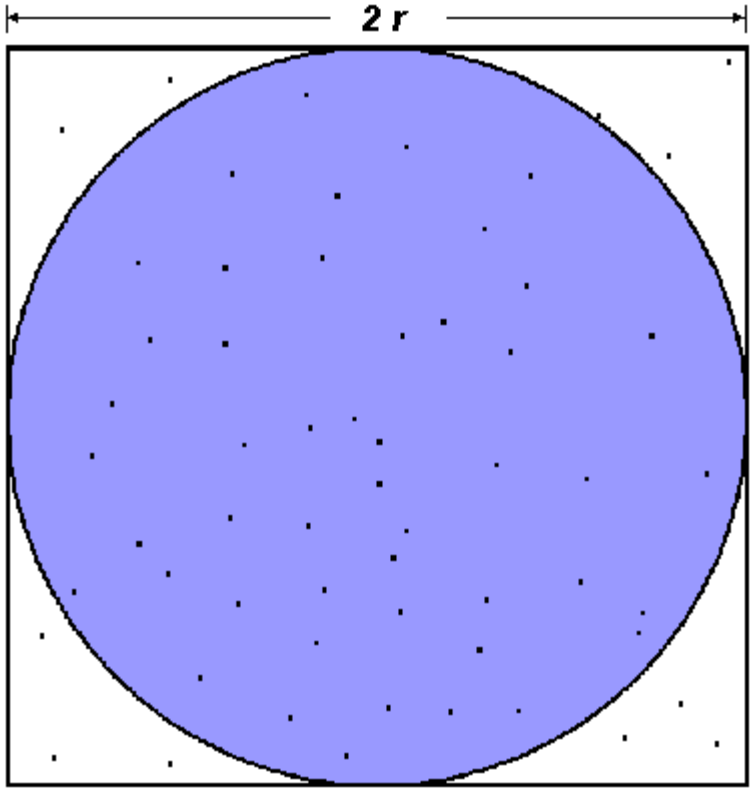
Primeri paralelnih programa

- Računanje broja PI
- Obrada 2D niza (array processing)
- *Simple heat equation* problem
- *1-D Wave Equation* problem

Računanje broja PI (1)

- Jedan jednostavan algoritam za približno računanje broja PI
  - Upisati krug u kvadrat
  - Slučajno generisati tačke u datom kvadratu
  - Utvrditi broj tačaka u kvadratu koje su istovremeno i u krugu
  - PI se može približno izračunati po formuli
    - PI = 4.0 * (broj tačaka u krugu)/(ukupan broj tačaka)
  - Što se više tačaka generiše, aproksimacija je bolja

Računanje broja PI (2)



$$A_S = (2r)^2 = 4r^2$$

$$A_C = \pi r^2$$

$$\pi = 4 \times \frac{A_C}{A_S}$$

Računanje broja PI (3)

- Sekvencijalni kod
  - npoints = 10000
    circle_count = 0
    do j = 1,npoints
      generate 2 random numbers between 0 and 1
      xcoordinate = random1 ; ycoordinate = random2
      if (xcoordinate, ycoordinate) inside circle
      then circle_count = circle_count + 1
    end do

    PI = 4.0*circle_count/npoints

Računanje broja PI (4)

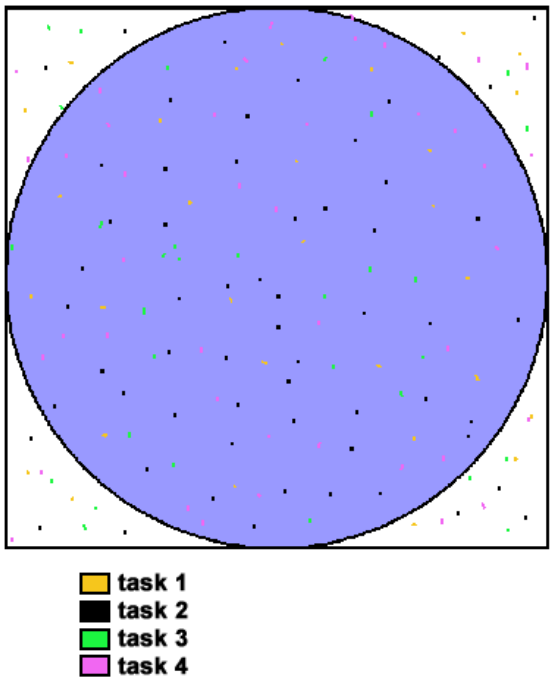
- Većina vremena u izvršavanju ovog koda bi bila potrošena na izvršavanje petlje
- Još jedan od *embarrassingly parallel* problema
  - Računski zahtevan, minimalna komunikacija i I/O
- Paralelna strategija
  - Razbiti petlju na delove koje će izvršavati zadaci
  - Svaki zadatak obavlja svoj deo računanja petlje
  - Ne postoje zavisnosti po podacima, pa nema ni komunikacije među zadacima
  - Koristi se SPMD model
    - Jedan zadatak se ponaša kao master i sakuplja rezultate

Računanje broja PI (5)

- npoints = 10000
  circle_count = 0
  p = number of tasks
  num = npoints/p
  find out if I am MASTER or WORKER
  do j = 1,num
    generate 2 random numbers between 0 and 1
    xcoordinate = random1 ; ycoordinate = random2
    if (xcoordinate, ycoordinate) inside circle
    then circle_count = circle_count + 1
  end do

Računanje broja PI (6)

- if I am MASTER
  receive from WORKERS their circle_counts
  compute PI (use MASTER and WORKER calculations)
  else if I am WORKER
  send to MASTER circle_count
  endif

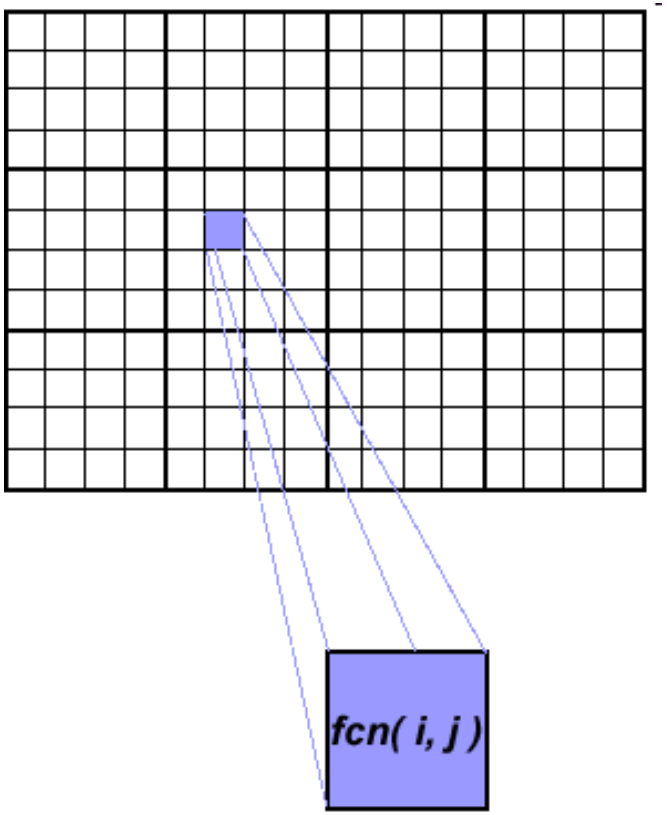


Obrada 2D niza (1)

- Radi se obrada nad elementima 2D niza, a računski posao nad svakim elementom je nezavisan u odnosu na druge
- Sekvencijalni program računa jedan element u jednom trenutku idući sekvencijalno kroz niz
- Ceo problem je računski veoma zahtevan
- Problem je tzv. *embarrassingly parallel*, jer je izračunavanje jednog elementa nezavisno od drugih elemenata

Obrada 2D niza (2)

- Sekvencijalno rešenje
  - do j = 1,n
    do i = 1,n
      a(i,j) = fcn(i,j)
    end do
    end do

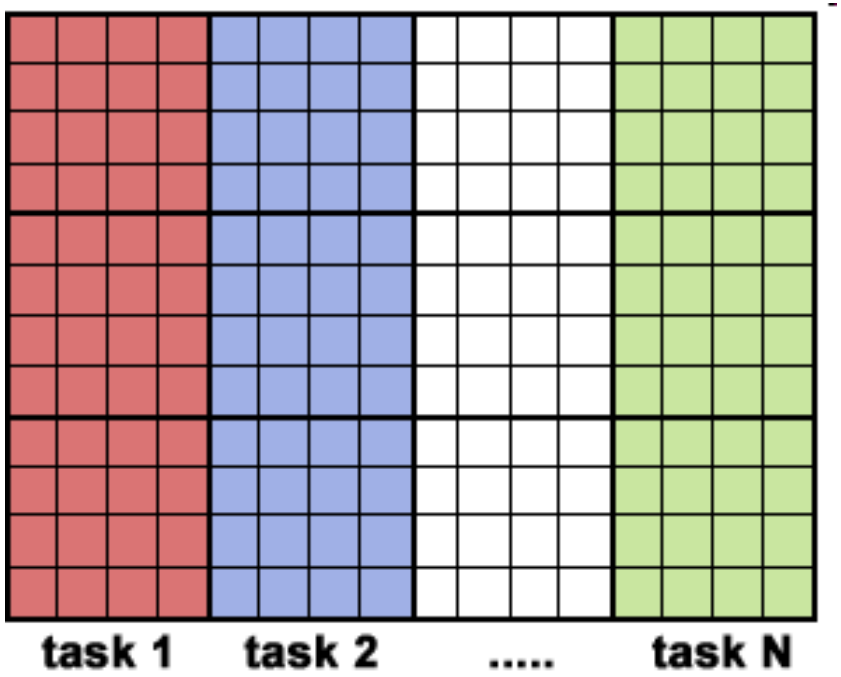


Obrada 2D niza (3)

- Paralelno rešenje
  - Elementi niza se podele tako da svaki procesor dobije jednu podmatricu na obradu
  - Prilikom izbora šeme za distribuciju podmatrica, gleda se da maksimizuje korišćenje keš memorije
  - Komunikacija između zadataka nije potrebna zbog karakteristika problema
  - Nakon što se matrica distribuira, svaki zadatak izvršava deo petlje, u zavisnosti koji deo podataka je dobio na obradu

Obrada 2D niza (4)

- Primer koda
  - do j = mystart, myend
    do i = 1,n
      a(i,j) = fcn(i,j)
    end do
    end do



Obrada 2D niza (5)

- Jedno moguće rešenje
  - Implementirati SPMD model
  - Master zadatak inicijalizuje niz,
    šalje podatke radnicima i prima rezultate
  - Zadatak-radnik prima podatke,
    izvršava svoj deo računanja i šalje rezultate masteru
  - Ovakvo rešenje se može oblikovati
    prema pseudo-kodu datom na sledećoj stranici

Obrada 2D niza (6)

- find out if I am MASTER or WORKER

  if I am MASTER

    initialize the array
    send each WORKER info on part of array it owns
    send each WORKER its portion of initial array

  receive from each WORKER results

Obrada 2D niza (7)

- else if I am WORKER
  receive from MASTER info on part of array I own
  receive from MASTER my portion of initial array

  # calculate my portion of array
  do j = my first column,my last column
  do i = 1,n
    a(i,j) = fcn(i,j)
  end do
  end do

  send MASTER results

  endif

Obrada 2D niza (8)

- Prethodno rešenje prikazuje statičko balansiranje opterećenja
  - Svaki zadatak dobija podjednaku količinu posla
  - Ukoliko su procesori različitih karakteristika, može doći do gubitka procesorskog vremena u čekanju
  - Statičko balansiranje opterećenje obično nije problem ukoliko zadaci izvršavaju podjednak posao na identičnim mašinama
  - Ukoliko postoji problem balansiranosti opterećenja, može se koristiti šema bazena poslova (*pool of tasks*) za rešavanje problema

Obrada 2D niza (9)

- Depo poslova (*task pool*)
  - Postoje dve vrste procesa: master i proces-radnik
- Master proces
  - Sadrži i održava depo poslova koje radnici treba da obave
  - Šalje radniku posao kada to ovaj zahteva
  - Sakuplja rezultate od procesa-radnika
  - Po potrebi, i master može da se uključi u obavljanje posla

Obrada 2D niza (10)

- Proces-radnik
  - Dobija poslove od master procesa
  - Izvršava računanje
  - Šalje rezultate master procesu
- Proces-radnik ne zna pre vremena izvršavanja koji deo posla će obaviti i koliko poslova
- Dinamičko balansiranje opterećenja se dešava u vreme izvršavanja
  - Brži procesi-radnici dobijaju više posla da obave

Obrada 2D niza (11)

- find out if I am MASTER or WORKER

  if I am MASTER

    do until no more jobs
      send to WORKER next job
      receive results from WORKER
    end do

    tell WORKER no more jobs

Obrada 2D niza (12)

- else if I am WORKER

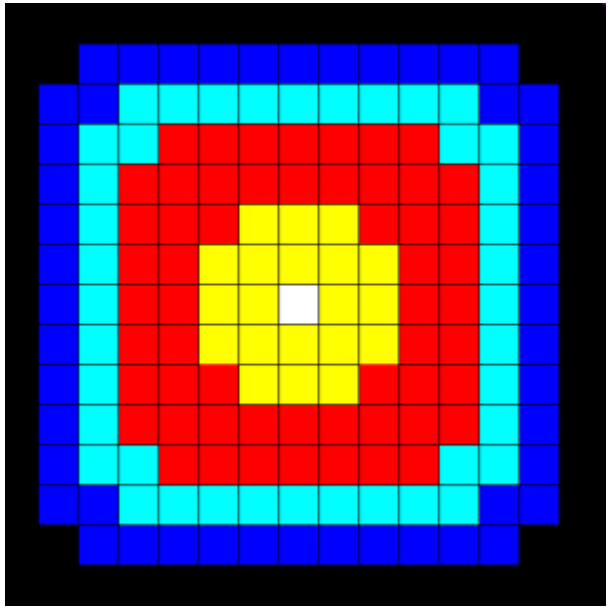
```
  do until no more jobs

    receive from MASTER next job
       calculate array element: a(i,j) = fcn(i,j)
    send results to MASTER

  end do
endif
```

Obrada 2D niza (13)

- Depo poslova
  - U datom primeru, svaki zadatak je smatrao jedan element niza za poseban posao (*job*)
  - Primer *fine-grain* implementacije
  - *Fine-grain* implementacije podrazumevaju više režijskog vremena za komunikaciju, kako bi se smanjilo rasipanje procesorskog vremena
  - Optimalnije rešenje bi bilo da svaki posao koji se šalje sadrži veću količinu računskog posla
    - Količina posla zavisi od konkretnog problema

Simple heat equation problem (1)

- Većina paralelnih problema zahteva komunikaciju među zadacima
  - Postoji čitava klasa problema koji zahtevaju komunikaciju sa "susednim" zadacima
- *Simple heat equation* problem opisuje promenu temperature na nekom prostoru kroz vreme, kada su poznati početna temperatura i granični uslovi



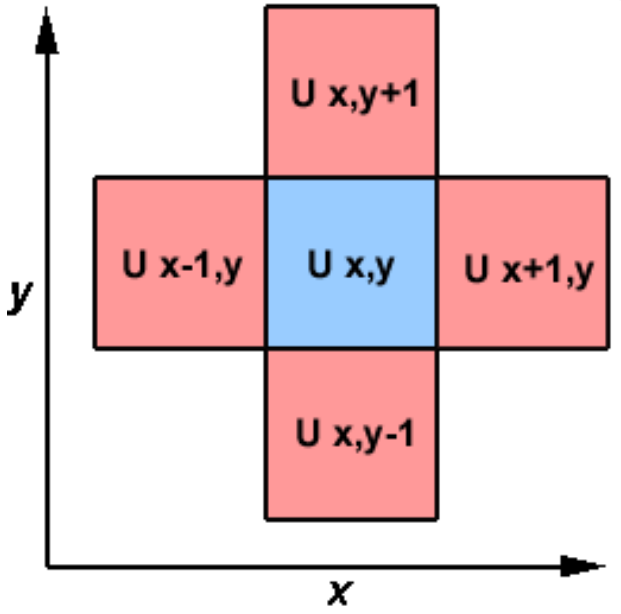
Simple heat equation problem (2)

- Postoji konačna šema za rešavanje ovog problema numerički na kvadratnoj površini
  - Početna temperatura na granicama je nula, a u sredini je visoka
  - Granična temperatura se uvek drži na nuli
  - Kvadratna površina se deli na mrežu manjih kvadrata
    - Predstavljaju se matricom čiji elementi predstavljaju temperaturu odgovarajućeg polja u kvadratu
    - Koristi se *time stepping* algoritam – izračunaju se temperature svih polja u datom vremenskom trenutku, pa se prelazi na sledeći vremenski trenutak

Simple heat equation problem (3)

- Izračunavanje vrednosti jednog elementa zavisi od vrednosti susednih elemenata

$$U_{x,y} = U_{x,y} + C_x * (U_{x+1,y} + U_{x-1,y} - 2 * U_{xy}) + C_y * (U_{x,y+1} + U_{x,y-1} - 2 * U_{x,y})$$



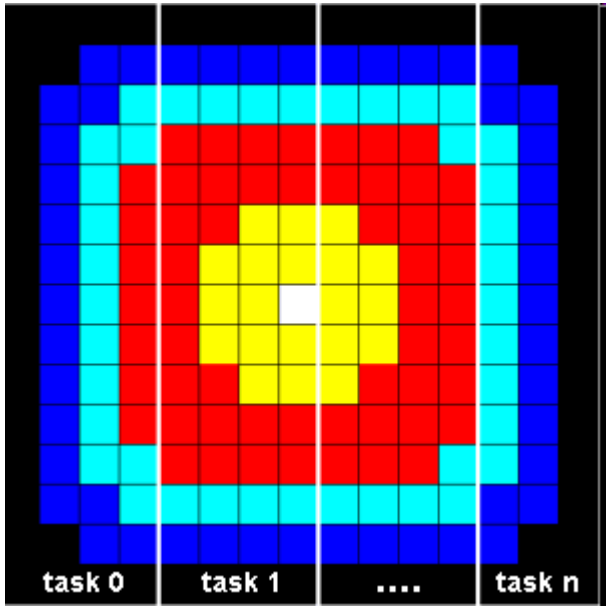
Simple heat equation problem (4)

- Sekvencijalno rešenje
  - do iy = 2, ny – 1
    do ix = 2, nx – 1

```
u2(ix, iy) =
  u1(ix, iy)  +
    cx * (u1(ix+1,iy) + u1(ix-1,iy) - 2.*u1(ix,iy)) +
    cy * (u1(ix,iy+1) + u1(ix,iy-1) - 2.*u1(ix,iy))

end do
end do
```

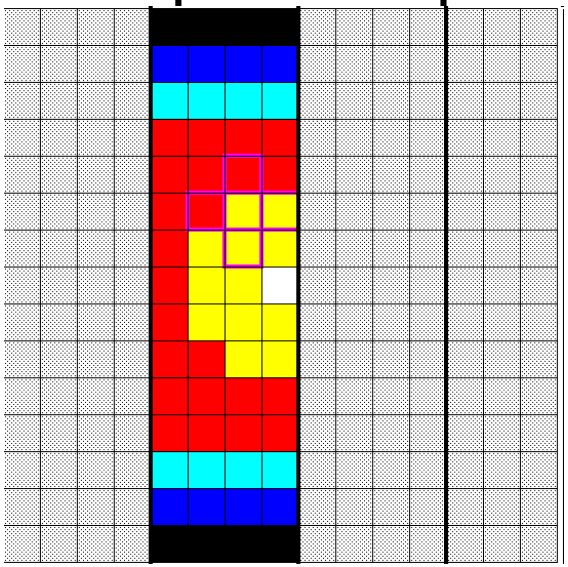
Simple heat equation problem (5)

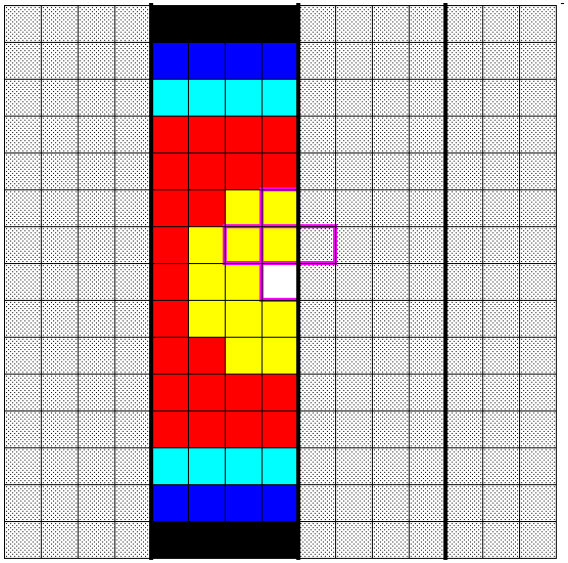
- Jedno paralelno rešenje
  - Implementirati SPMD model
  - Cela matrica se deli na i svaki zadatak dobija deo posla
  - Master proces šalje podatke procesima-radnicima, proverava konvergenciju rešenja i sakuplja rezultate
  - Procesi-radnici računaju podatke, komuniciraju po potrebi sa susednim procesima i šalju rezultate masteru



Simple heat equation problem (6)

- Postoje zavisnosti po podacima
  - Unutrašnji elementi koji pripadaju zadatku ne zavise od drugih zadataka
  - Granični elementi zahtevaju komunikaciju, jer su im potrebni podaci iz susednih zadataka





Simple heat equation problem (7)

- find out if I am MASTER or WORKER
  if I am MASTER
    initialize array
    send each WORKER starting info and subarray
    do until all WORKERS converge
      gather from all WORKERS convergence data
      broadcast to all WORKERS convergence signal
    end do
    receive results from each WORKER

Simple heat equation problem (8)

- else if I am WORKER
  receive from MASTER starting info and subarray
  do until solution converged
    update time
    send neighbors my border info
    receive from neighbors their border info
      update my portion of solution array
    determine if my solution has converged
      send MASTER convergence data
      receive from MASTER convergence signal
  end do
  send MASTER results
  endif

Simple heat equation problem (9)

- U prethodnom rešenju je pretpostavljeno da je komunikacija između zadataka blokirajuća
  - Blokirajuća komunikacija podrazumeva da zadatak ne nastavlja dalje sa izvršavanjem dok god primalac ne preuzme podatke
- U prethodnom rešenju susedni zadaci razmene podatke, a zatim svaki od njih izračuna svoj deo matrice
- Vreme računanja se može često smanjiti korišćenjem neblokirajuće komunikacije
  - Neblokirajuća komunikacija dozvoljava da se nastavi sa radom dok komunikacija još traje
  - Svaki zadatak može da izračuna svoje unutrašnje elemente dok se vrši razmena podataka o graničnim elementima, a zatim da izračuna vrednost svojih graničnih elemenata

Simple heat equation problem (10)

- find out if I am MASTER or WORKER
  if I am MASTER
    initialize array
    send each WORKER starting info and subarray
    do until all WORKERS converge
      gather from all WORKERS convergence data
      broadcast to all WORKERS convergence signal
    end do
    receive results from each WORKER

Simple heat equation problem (11)

- else if I am WORKER
  receive from MASTER starting info and subarray
  do until solution converged
    update time
    non-blocking send neighbors my border info
    non-blocking receive neighbors border info
    update interior of my portion of solution array
    wait for non-blocking communication complete
    update border of my portion of solution array

Simple heat equation problem (12)

- determine if my solution has converged
  send MASTER convergence data
  receive from MASTER convergence signal

  end do
  send MASTER results
  endif

1-D Wave Equation problem (1)

- Treba izračunati amplitudu talasa
  duž uniformne, vibrirajuće žice
  nakon odgovarajućeg vremenskog trenutka
- Računanje uključuje
  - Vrednost amplitude talasa na *y* osi
  - Poziciju tačke koja se prati duž *x* ose
  - Presečne tačke talasa duž žice
  - Promenu amplitude talasa
    u diskretnim vremenskim intervalima

1-D Wave Equation problem (2)

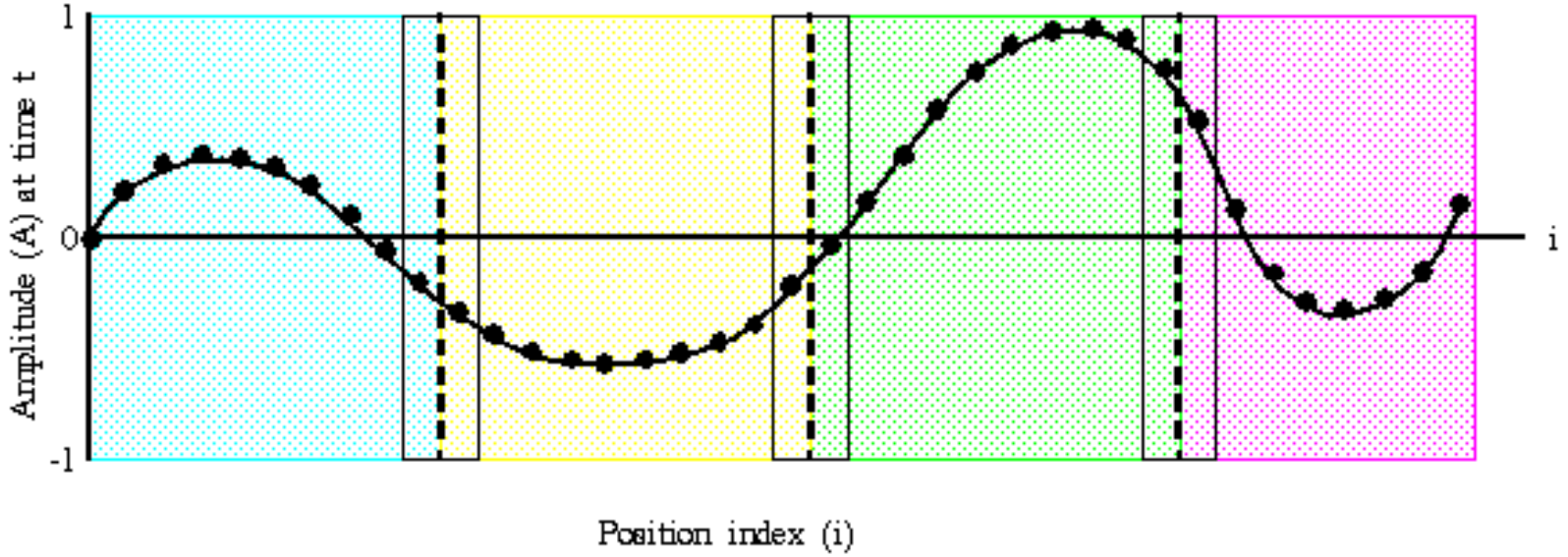
- Amplituda talasa u jednoj tački se izračunava po formuli
  - A(i,t+1) =
    (2.0 * A(i,t)) - A(i,t-1)+ (c * (A(i-1,t) - (2.0 * A(i,t)) + A(i+1,t)))
- Amplituda talasa u nekoj tački zavisi
  i od susednih tačaka
  i od amplitude tačke u prethodnim vremenskim trenucima
  - Paralelno rešenje će zahtevati komunikaciju

1-D Wave Equation problem (3)

- Moguće rešenje
  - Implementirati SPMD model
  - Talas predstaviti pomoću niza amplituda u izabranim tačkama
  - Niz amplituda podjednako podeliti i poslati podnizove svim zadacima koji rade
  - Obrada svake tačke zahteva podjednako vreme, tako da svi zadaci treba da dobiju jednak broj tačaka
  - Obratiti pažnju da zadaci dobiju što veći kontinualni deo niza, kako bi se izbegla nepotrebna komunikacija

1-D Wave Equation problem (4)

- Komunikacija potrebna na granicama podnizova
  - "susedni" zadaci će morati da razmenjuju podatke



1-D Wave Equation problem (5)

- find out number of tasks and task identities

```
#Identify left and right neighbors
left_neighbor = mytaskid – 1
right_neighbor = mytaskid +1
if mytaskid = first then left_neigbor = last
if mytaskid = last then right_neighbor = first

find out if I am MASTER or WORKER
if I am MASTER
  initialize array
  send each WORKER starting info and subarray
else if I am WORKER
  receive starting info and subarray from MASTER
endif
```

1-D Wave Equation problem (6)

- #Update values for each point along string

```
#In this example the master participates in calculations
do t = 1, nsteps
  send left endpoint to left neighbor
  receive left endpoint from right neighbor
  send right endpoint to right neighbor
  receive right endpoint from left neighbor

#Update points along line
  do i = 1, npoints
    newval(i) = (2.0 * values(i)) - oldval(i)
    + (sqtau * (values(i-1) - (2.0 * values(i)) + values(i+1)))
  end do
end do
```

1-D Wave Equation problem (7)

- #Collect results and write to file
  if I am MASTER
    receive results from each WORKER
    write results to file
  else if I am WORKER
    send results to MASTER
  endif

Opšta terminologija

Opšta terminologija (1)

- Zadatak (*task*)
  - Logički diskretan deo posla, tipično program ili skup instrukcija koje izvršava procesor
- Paralelni zadatak (*parallel task*)
  - Zadatak koji se bezbedno može izvršiti na multiprocesorskom sistemu (daje korektne rezultate)
- Sekvencijalno izvršavanje (*serial execution*)
  - Izvršavanje programa sekvencijalno, jedna instrukcija u jednom trenutku
  - U svim paralelnim programima postoje sekcije koje se moraju izvršiti sekvencijalno

Opšta terminologija (2)

- Paralelno izvršavanje (*parallel execution*)
  - Izvršavanje programa u vidu više zadataka koji mogu da izvršavaju različite instrukcije u istom trenutku
- Deljena memorija (*shared memory*)
  - U hardverskom smislu, to je računarska arhitektura kod koje svi procesori imaju direktan pristup zajedničkoj, deljenoj fizičkoj memoriji
  - U programskom smislu, to je model kod koga svi paralelni zadaci imaju istu "sliku" memorije i mogu direktno da pristupe istim logičkim memorijskim lokacijama gde god se fizička memorija stvarno nalazi

Opšta terminologija (3)

- Distribuirana memorija (*distributed memory*)
  - Hardverski, model zasnovan na pristupu fizičkoj memoriji putem mreže
  - Programski, zadaci logički vide samo podatke iz svog adresnog prostora i za razmenu podataka sa drugim zadacima koriste komunikacione primitive
- Komunikacija (*communication*)
  - Paralelni zadaci imaju potrebu da razmenjuju podatke
  - Više načina da se to ostvari
  - Ovaj termin se upotrebljava da označi razmenu podataka bez obzira na način na koji se to radi

Opšta terminologija (4)

- Sinhronizacija (*synchronization*)
  - Koordinacija paralelnih zadataka u vremenu izvršavanja, po pravilu u vezi sa komunikacijom
  - Često se implementira kao tačka u programu koju zadatak ne sme da pređe pre nego što drugi zadaci dostignu logički ekvivalentnu tačku (barijera)
  - Sinhronizacija obično uključuje stanje čekanja bar jednog zadatka, i stoga produžava vreme izvršavanja paralelne aplikacije

Opšta terminologija (5)

- Granularnost (*granularity*)
  - U paralelnom računarstvu, to je kvalitativna mera odnosa korisnog rada (računskog posla) i komunikacije
  - Gruba (*coarse grain*) granularnost
    - Relativno velika količina računskog posla se obavi između dve komunikacije
  - Fina (*fine grain*) granularnost
    - Relativno mala količina računskog posla se obavi između dve komunikacije

Opšta terminologija (6)

- Primećeno ubrzanje (*observed speedup*)
  - Odnosi se na ubrzanje dobijeno paralelizacijom nekog sekvencijalnog koda
  - Definiše se kao odnos vremena sekvencijalne i paralelizovane aplikacije
  - Jedan od najjednostavnijih i najkorišćenijih indikatora performansi paralelnog programa
- Masovno paralelan (*massively parallel*)
  - Odnosi se na hardver koji obuhvata mnogo procesora
  - Značenje reči "mnogo" raste, a trenutno se može opisati sa 5 ili 6 cifara

Opšta terminologija (7)

- Režijsko vreme paralelizacije (*parallel overhead*)
  - Količina vremena
    potrebna da se koordinišu paralelni zadaci
  - Uključuje faktore kao što su vreme pokretanja i završetka, sinhronizacija, komunikacija,
    utrošak vremena zbog paralelnih prevodilaca, biblioteka, alata, operativnih sistema
- Klaster računarstvo (*cluster computing*)
  - Upotreba uobičajenih komponenti (procesora, mrežnih komponenti i slično) za izgradnju paralelnog sistema
- Računarstvo visokih performansi (*supercomputing / high performance computing*)
  - Upotreba najmoćnijih računara za rešavanje velikih problema

Opšta terminologija (8)

- Višejezgarni procesori (*multi-core processors*)
  - Procesori sa više jezgara na istom čipu
- Skalabilnost (*scalability*)
  - Odnosi se na mogućnost paralelnog sistema da proporcionalno ubrza svoj rad sa dodavanjem novih procesora (resursa) u sistem
  - Faktori koji utiču na skalabilnost
    - Hardver, naročito propusni opseg procesora i memorije i mrežna komunikacija
    - Algoritam i karakteristike same paralelne aplikacije
    - Režijsko vreme paralelizacije (*parallel overhead*)

Opšta terminologija (9)

- Kašnjenje (*latency*)
  - Vreme potrebno da se prazna poruka prenese iz tačke A u tačku B i najčešće se izražava u mikrosekundama
- Propusni opseg (*bandwidth*)
  - Predstavlja količinu podataka koja može da se prenese u jedinici vremena i izražava se u MB/s

Izvori o paralelnom programiranju

Literatura i izvori

- Blaise Barney,
  Introduction to Parallel Computing,
  https://computing.llnl.gov/tutorials/parallel_comp/,
  2016.
- Ian Foster,
  Designing and Building Parallel Programs
- Grama, Gupta, Karypis, Kumar,
  Introduction to Parallel Computing
- Culler, Singh, Gupta,
  Parallel Computer Architecture
  (A Hardware/Software Approach), 1998.