



Fakulteta za
informacijske študije
Faculty of information studies

Univerza v Ljubljani
Fakulteta *za strojništvo*



Kreativno jedro:
Simulacije
Creative core: Simulations

Kampus šola HPC 2014

Vzporedni algoritmi (snovanje in analiza)

Matjaž Depolli
matjaz.depolti@ijs.si
Odsek za komunikacijske sisteme
Inštitut „Jožef Stefan“



Naložba v vašo prihodnost
OPERACIJO DELNO FINANCIRA EVROPSKA UNIJA
Evropski sklad za regionalni razvoj



REPUBLIKA SLOVENIJA
**MINISTRSTVO ZA IZOBRAŽEVANJE,
ZNANOST IN ŠPORT**

Pregled predavanja

- Flynnova taksonomija in tipi vzporednosti
- Pohitritev in učinkovitost
- Nivoji vzporednosti v strojni opremi
- Izkoriščanje vzporednosti v programih
- Snovanje vzporednih algoritmov
 - Iskane lastnosti
 - Faze procesa snovanja

Flynnova taksonomija (1966)

- 4 skupine računalnikov:
 - SISD (Single Instruction Single Data)
 - SIMD (Single Instruction Multiple Data)
 - MISD (Multiple Instruction Single Data)
 - MIMD (Multiple Instruction Multiple Data)
- SISD = Von Neumannov model (zaporedni)
- Ostalo so paralelni modeli

Primeri

- SIMD:
 - GPU
 - Vektorske operacije (inštrukcije): MMX, SSE, ...
- MISD:
 - bi bili primerni za iskanje praštevil
 - Operacija 'sincos' na x86 arhitekturi
- MIMD
 - Superskalarna arhitektura
 - Multi- in many-core arhitekture
 - Lahko simulira SISD, SIMD in MIMD.

Tipi vzporednosti

- Podatkovna vzporednost
 - 'Data parallelism'
 - $F(X) = [F(x_1), F(x_2), F(x_3), \dots, F(x_n)]$; pri $X = [x_1, x_2, x_3, \dots, x_n]$
 - Vsak od procesorjev dobi v izračun podmnožico vseh podatkov
- Postopkovna vzporednost
 - 'Task parallelism'
 - $F(X) = F_1(F_2(F_3(\dots(F_n(X))\dots)))$; pri $F = F_1 F_2 F_3 \dots F_n$
 - Cevocod, procesor i dobi v izračun funkcijo F_i
- Asinhrona vzporednost in ostale eksotične oblike

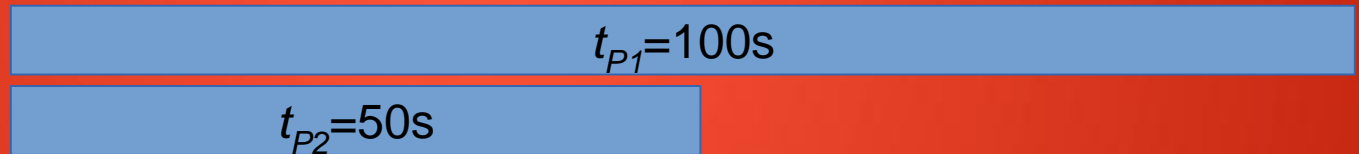
Pohitritev in učinkovitost

- Pohitritev (speedup)
 - Pohitritev programa P1 v primerjavi s P2: $S = t_{p1}/t_{p2}$
 - Vzporedna pohitritev na n procesorjih: $S = t_1/t_n$
 - Primer: P1 rabi za izračun 10 s, P2 rabi za ekvivalenten izračun 5 s.

$$t_{P1} = 10 \text{ s}$$

$$t_{P2} = 5 \text{ s}$$

$$S = 10/5 = 2$$



- Pohitritev vzporednega programa (absolutna) se vedno računa relativno na najhitrejši zaporedni algoritem
- Pohitritev je motivacija za paralelnost in za HPC

Pohitritev in učinkovitost

- Učinkovitost (efficiency) vzporednega algoritma na n procesorjih:
 - $E = S/n$ ($= t_1/n * t_n$)
 - Prejšnji primer s tem da se P1 izvaja na enem procesorju, P2 pa na dveh:
 $E = 2/2 = 1$
- Učinkovitost 1 (pohitritev enaka n) je težko dosegljiva
 - Linearna pohitritev
- Učinkovitost nad 1 (pohitritev večja od n) je možna
 - Super-linearna pohitritev

V razmislek

- Najvišja pohitritev je odvisna od problema
 - Povsem zaporedni problemi (Newtonova metoda)
 - T.i. 'embarasingly parallel' problemi
- Teoretična in praktična pohitritev
- Odvisna je od računalnika
 - S čim primerjamo pohitritev na 200 jedrih GPUja?
- Lahko je delno naključna
 - Stohastični algoritmi, delno naključna izvedba na vzporednem računalniku
- Super-linearna pohitritev včasih nakazuje neoptimalen zaporedni algoritem

Omejitve pohitritve

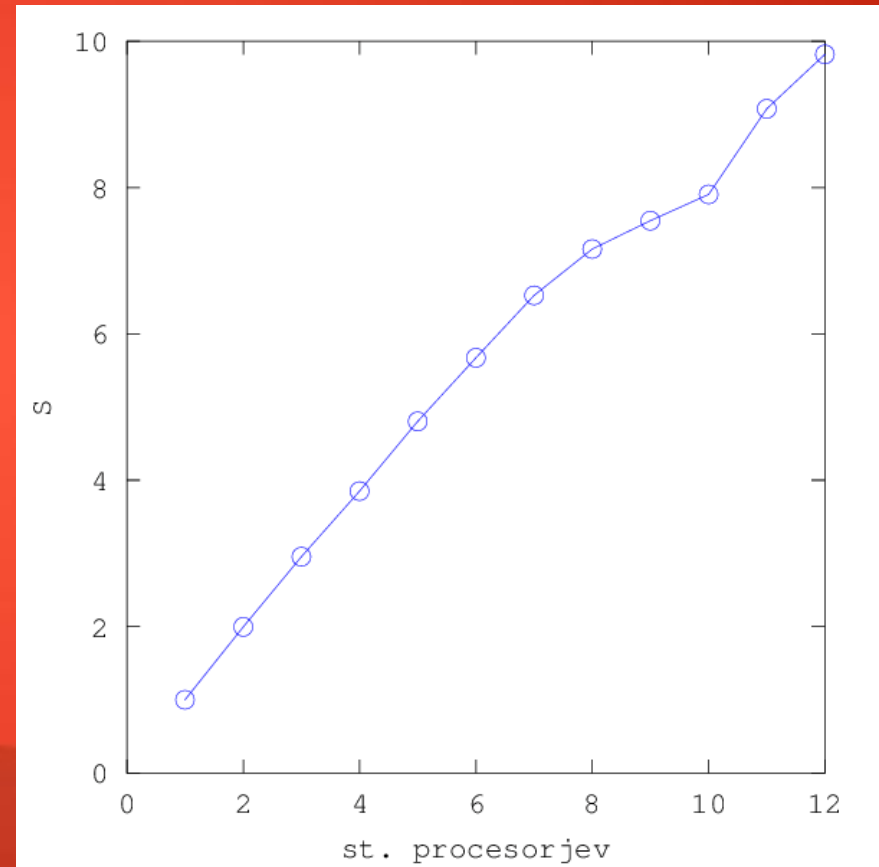
- Zaporedna (sekvenčna) narava problema
 - Delež problema, ki se mora izvesti zaporedno
 - Amdahlov zakon: minimalni čas za rešitev problema je $t_s + t_p / n$
 - Z večanjem n se približujemo času t_s , ki je absolutna meja
- Izgube zaradi paralelizacije (overhead)
 - Dekompozicija problema
 - Pošiljanje sporočil
 - Upravljanje z nitmi
 - Uporaba oddaljenega pomnilnika
 - Sinhronizacija

Primeri

Iskanje največje klike

Prvi tip vhodnega grafa

- Slika kaže na razmeroma linearno pohitritev
- Nihanja zato, ker niha število korakov iskanja

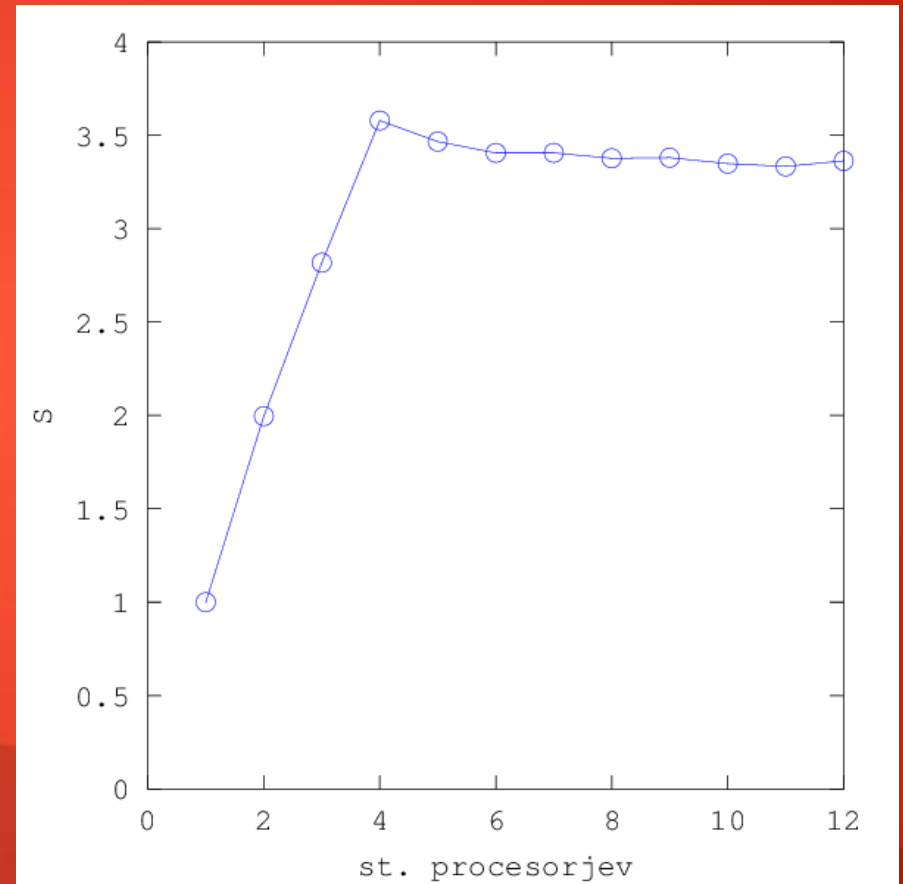


Primeri

Iskanje največje klike

Drug tip vhodnega grafa

- Več kot 4 procesorji na enkrat ne koristijo; celo škodujejo

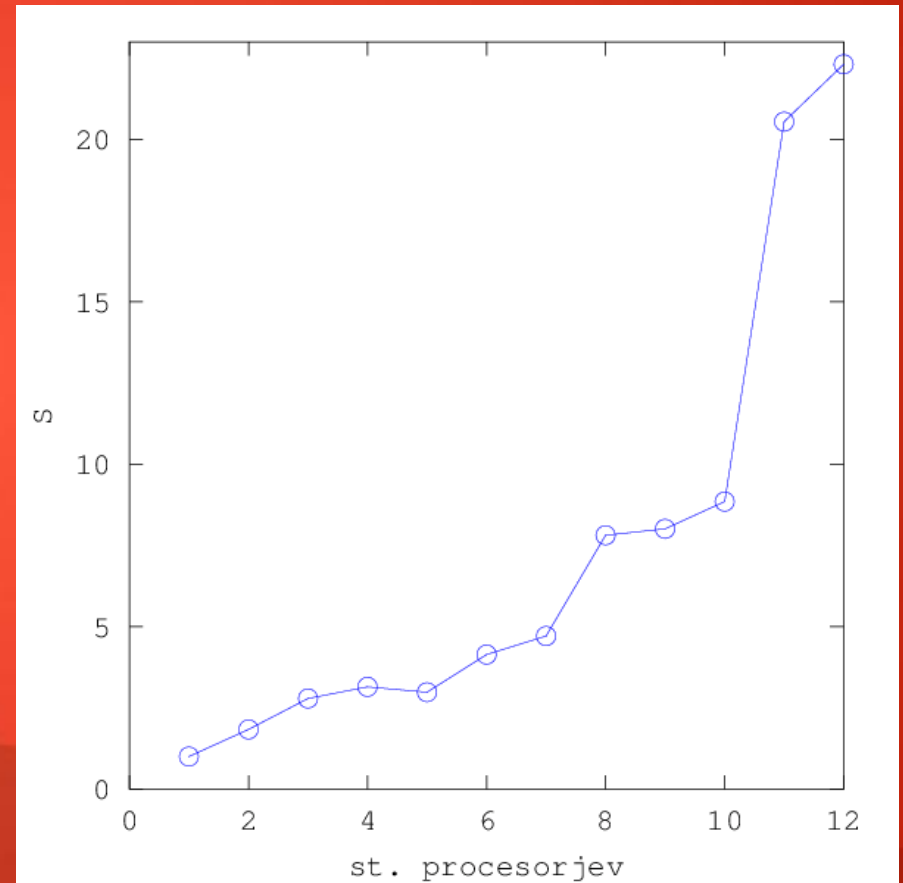


Primeri

Iskanje največje klike

Tretji tip vhodnega grafa

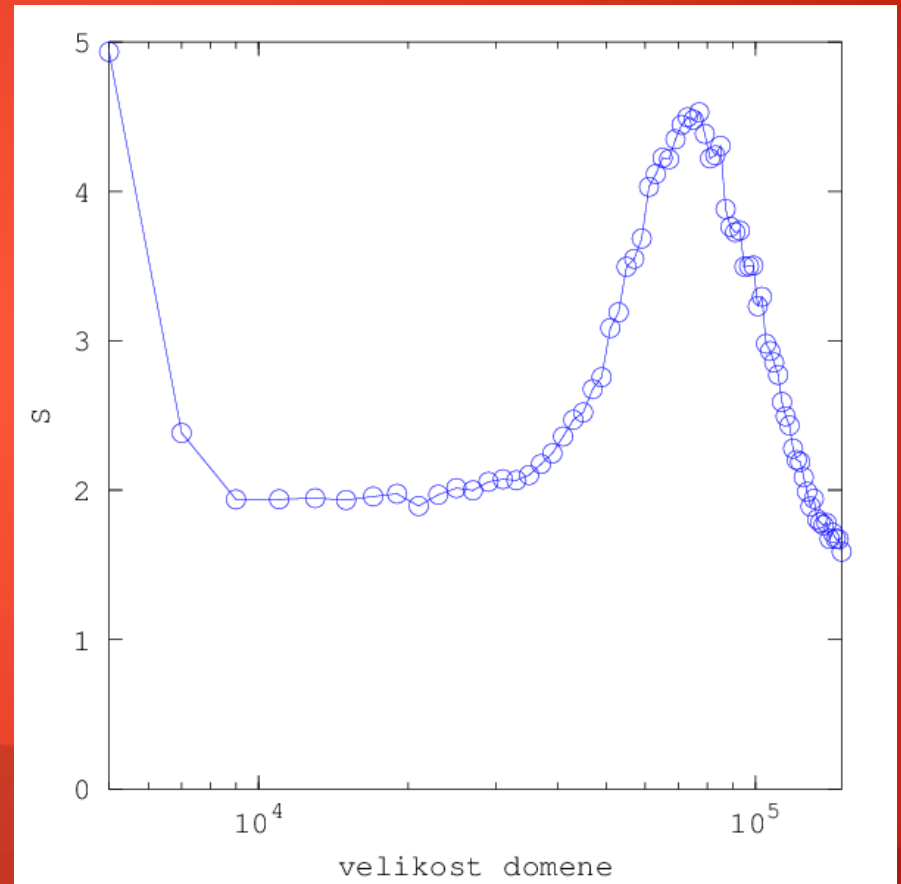
- Hitrost preiskovanja drevesa rešitev je odvisna tudi od sreče
- Pohitritev zaradi manj opravljenega dela



Primeri

Simulacija difuzije
temperature na 2D
domeni

- Slika pohitritve na 2 procesorjih
- Super-linearna pohitritev ko velikost domene sovpada z velikostjo dveh predpomnilnikov



Nivoji uvedbe vzporednosti

- Bitni: operacije nad vektorji bitov
 - Operacije nad 8-bitnimi operandi na več bitni arhitekturi
- Na nivoju operacij: cevovod
- Vektorske operacije: po 2, 4, 8, 16 operandov naenkrat
- Super-skalarnost: več neodvisnih operacij se izvede hkrati
- Multi- in many- core arhitekture: skupen pomnilnik
- Porazdeljeno računanje: komunikacija prek sporočil

Izkoriščanje vzporednosti

- Avtomatsko
 - Paralelni programski jeziki (Erlang, OpenCL)
 - Porazdeljene podatkovne baze
 - Porazdeljeni datotečni sistemi
- Visoko-nivojsko
 - MapReduce
- Srednje-nivojsko
 - MPI
 - OpenMP
- Direktno / nizko-nivojsko
 - Uporaba bitnih in vektorskih operacij, niti, procesov, ...

Pogoste težave pri vpeljavi vzporednosti

- Razdelitev na podprobleme, ki niso popolnoma neodvisni
 - Odvisnosti zahtevajo komunikacijo med procesorji, ki rešujejo podprobleme
- Podproblemi niso enako zahtevni
 - 'Load balancing'
 - Razdelitev na čim manjše podprobleme
- Zaporedni deli problema

Pomembni vzporedni algoritmi

- Linearna algebra
- FFT
- Mrežne in brez-mrežne metode
- Monte Carlo simulacije
- Algoritmi nad grafi in drevesi
- Dinamično programiranje
- Problem n teles
- Kombinatorijska logika (preslikave, šifriranje)

Amdahlov in Gustafsonov zakon

- Amdahl:

- Obstaja mejna pohitritev, ki se ji lahko približamo z veliko procesorji, ne moremo je pa preseči
- Vzrok leži v zaporednem delu algoritma
- Ni smiselno imeti zelo velikega števila procesorjev

- Gustafson:

- Zaporedni delež algoritma ponavadi ni konstanten ampak odvisen od velikosti problema; z velikostjo se zmanjšuje
- Pri zelo velikih problemih postane zaporedni del nepomemben
- Velike probleme se uspešno rešuje z veliko procesorji

Snovanje vzporednih algoritmov

- Identificirati je treba naslednje lastnosti vzporednosti
 - Sočasnosti: kaj se lahko izvaja hkrati
 - Skalabilnost: kako se lastnosti vzporednosti spreminjajo z večanjem problema in večanjem računalnika
 - Lokalnost
 - Modularnost

Sočasnost

- Deli programa se lahko izvajajo sočasno, če:
 - So na voljo vsi podatki
 - So reševanja posameznih delov neodvisna med seboj
- Primer: seštevanje števil od 1 do n :
 - Lahko razdelimo na $\log(n)$ neodvisnih korakov:
 - 1. korak: $a=1+2$; $b=3+4$; $c=4+5$; ... $z=n-1+n$;
 - 2. korak: $a=a+b$; $b=c+d$; $c=e+f$; ...
 - ...
 - $\log(n)$ -ti korak: $\text{rezultat}=a+b$;
 - Vsak korak potrebuje rezultate prejšnjega koraka
 - Zgodnji koraki potrebujejo zelo veliko seštevalnikov

Skalabilnost

- Učinkovitost programa se praviloma manjša s številom procesorjev
 - Povečuje se dodatno delo – drobljenje problema, komunikacija, ...
 - Manj kot se učinkovitost zmanjšuje, večja je skalabilnost
- Dobri (splošni) vzporedni algoritmi delujejo dobro ne glede na število procesorjev
- Skalabilnost omogoča enostaven prenos algoritma/programa na drug sistem
- Skalabilnost omogoča reševanje večjih problemov z močnejšim računalnikom

Lokalnost

- Je možnost, da program uporablja podatke iz lokalnega pomnilnika
 - To je ponavadi isti pomnilnik v katerem se nahaja tudi program na posameznem procesorju
- Dostopanje do lokalnega pomnilnika je običajno mnogo hitrejše (cenejše) od dostopanja do ostalih pomnilnikov
- Tipična razmerja latenc:
 - 1:10 med nivoji predpomnilnika
 - 1:100 L1 predpomnilnik proti RAMu
 - 1:100 RAM proti GB Ethernetu
 - 1:1000 RAM proti SSD disku

Modularnost

- Moduli so deli programa, ki jih lahko razvijamo ločeno med seboj
- Moduli se lahko združujejo ali uporabljajo za druge aplikacije
- Interakcija med moduli prek dobro definiranih vmesnikov
- Modularna zasnova zmanjšuje zapletenost in omogoča ponovno uporabo preizkušenih delov programa
- Modularnost preko kompozicije (organizacija modulov) pripomore k lokalnosti, skalabilnosti, prenosljivosti in sočasnosti

Snovanje vzporednega programa

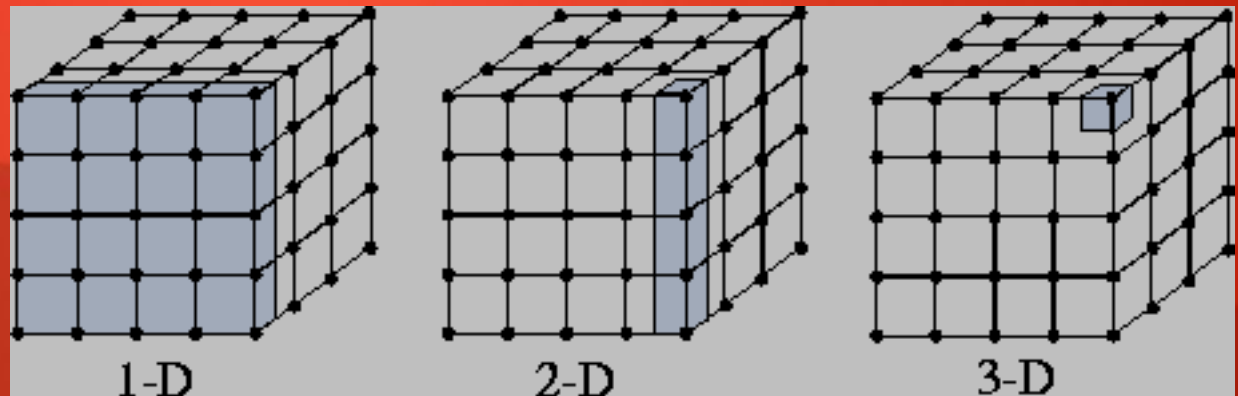
- Štiri faze (po Ianu Fosterju):
 - Delitev (partitioning)
 - Komunikacija (communication)
 - Združevanje (agglomeration)
 - Preslikava (mapping)
- Prvi dve fazi vplivata na sočasnost in skalabilnost
- Zadnji dve fazi poskušata uravnovežiti delo, najti čim več lokalnosti in izboljšati zmogljivosti programa na vzporednem računalniku

Delitev

- Cilj je razdelitev dela (obremenitve, opravil) med procesorje na vzporednem računalniku
- Rezultat naj bo čim večje število majhnih opravil, ki se lahko izvedejo sočasno
- Delitev po
 - Funkcijah – delijo se faze računanja
 - Domeni – delijo se podatki
- Delitev je abstraktna, splošna, ne deluje za ciljni računalnik

Delitev domene

- Primer – 3D kocka sestavljena iz 3D mreže točk.
- Možna je delitev v eni, dveh, ali treh dimenzijah
 - Razlike so v enostavnosti delitve in kasnejše komunikacije, fleksibilnosti rešitve, količini potrebne komunikacije
- Možna je tudi manj regularna delitev, ki pa pride bolj v poštev ob manj regularnih modelih

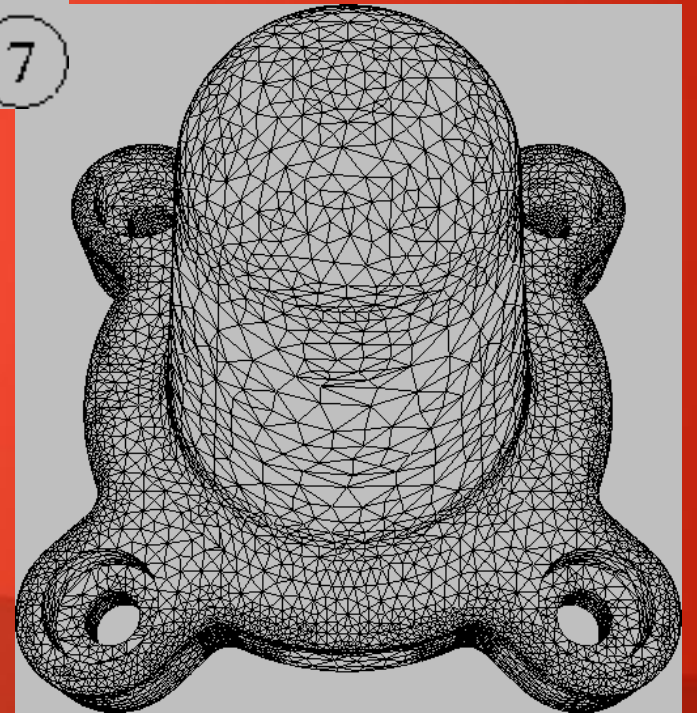
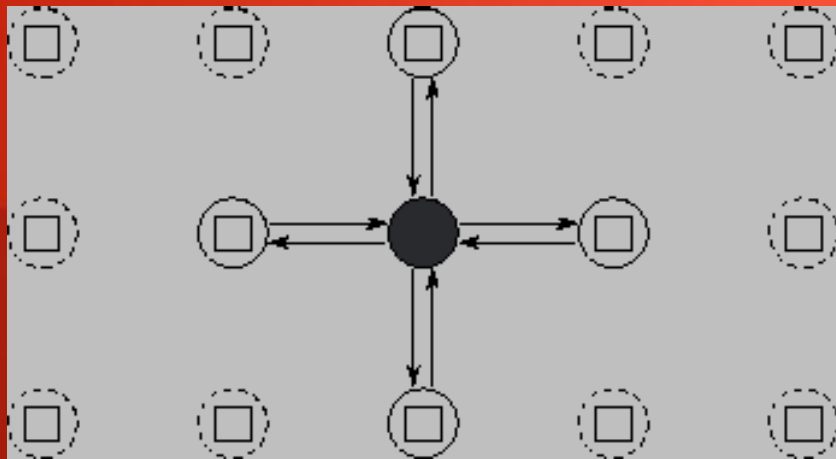
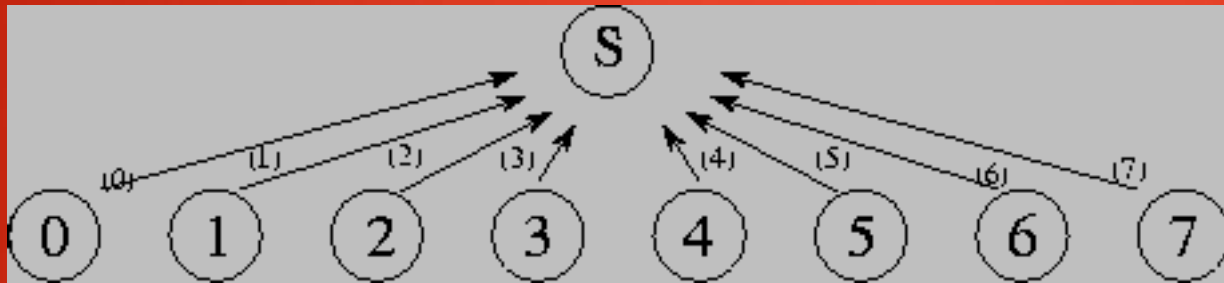


Komunikacija

- V splošnem se opravila, določena v fazi delitve, ne morejo izvajati popolnoma neodvisno, ampak vplivajo eden na drugega
- Obstajajo tudi 'globalni' podatki, ki se uporabljajo pri več opravilih
- Zasnova komunikacije opredeli kateri podatki se bodo prenašali in med katerimi opravili
 - Kanali – kje se bodo podatki prenašali
 - Izvori in ponori – kdo pošilja in kdo sprejema
 - Definicija sporočil

Komunikacija

- Delitev glede na pokritost opravil
 - Lokalna – opravilo komunicira le z majhno podmnožico ostalih opravil (s t.i. sosedi)
 - Globalna – opravilo komunicira nad veliko množico opravil ali celo z vsemi opravili
- Delitev glede na strukturo
 - Strukturirana – definira lokalno komunikacijo, ki se ne spreminja; npr. opravila povezuje v mrežo
 - Nestrukturirana – komunikacija je poljuben graf, lahko je odvisen od vhodnih podatkov, lahko se spreminja s časom



Združevanje

- Združevanje v nasprotju z delitvijo že cilja na določen vzporedni računalnik
- Cilji:
 - Zmanjšanje količine komunikacije z povečanjem zrnatosti
 - Ohranjanje skalabilnosti z zadosti velikim številom opravil (večjim od števila procesorjev)
 - Zmanjšanje cene programiranja – program naj bo čim bolj fleksibilen za bodoče aplikacije na drugačnih računalnikih

Preslikava

- Določa katero opravilo se bo izvedlo na kateri enoti (na izbiro so lahko procesorji in pospeševalniki)
- Lahko je statična ali dinamična – v slednjem primeru se določi le algoritem preslikave
- Cilja na uravnouteženo porazdelitev opravil med procesnimi enotami.
- Postopek uravnouteženja (load balancing) je lahko statičen (regularne mreže) ali pa se dinamično spreminja med izvajanjem programa



Povzetek

- Snovanje vzporednih algoritmov
 - Iskane lastnosti
 - Faze procesa snovanja
- Analiza vzporednih algoritmov
 - Pohitritev in učinkovitost
 - Skalabilnost
- Tipi vzporednosti v programih in njihovo izkoriščanje
- Nivoji vporednosti v strojni opremi

HiPEAC delavnica



High Performance and Embedded Architecture and Compilation

Integracija raziskovalcev, industrije in ponudnikov

25. 9. 2014; IJS, Ljubljana