Principer för design av parallella algoritmer

Grama et al.

Introduction to Parallel Computing
Kapitel 3

Mikael Rännar
(efter material av Robert Granat)

Grundläggande om parallell algoritmdesign

- Seriell algoritmdesign - sekvens av steg för att lösa givet problem m h a en dator, ett slags ”recept”
- Icketrivial parallell algoritmdesign - utökar seriella designen:
  - Identifiera delar av arbetet som kan utföras parallellt
  - Mappa parallella delar på flera processer som arbetar parallellt
  - Distribuera indata, resultat och mellanliggande värden
  - Hantera åtkomst till gemensamt eller delat data
  - Synkronisera processer under programexekveringens gång
- Framför allt:
  - Dela upp arbetet i mindre delar
  - Tilldela dessa delar till olika processorer
  - Annat kan falla sig naturligt längs vägen...

Begrepp för denna föreläsning

- Dekomposition
- Uppgifter (bättre: eng. *tasks*)
- Beroendegrafer
- Mappning
- Metoder för att dölja interaktion
- Modeller för parallella algoritmer

Detaljerad översikt

Dekomposition (uppdelning)
- Rekursion
- Data
- Undersökande
- Spekulativ
- Hybrid

Arbetsuppgifter (tasks)
- Egenskaper
- Egenskaper för interaktion mellan uppgifter

Beroendegrafer

Mappning
- Statisk
- Dynamisk

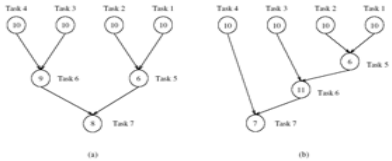
Metoder för att dölja interaktion
- Maximera lokalitet
- Minimera flaskhalsar
- Överlappa beräkningar och kommunikation
- Kollektiva kommunikations-operationer
- Överlappa interaktioner
- Replikera data
- Extra beräkningar

Modeller för parallella algoritmer
- Dataparallell
- Uppgiftsgraf
- Arbetspool
- Master-slave
- Pipe-line

Dekomposition – olika typer av uppdelning

- Lösa ett problem parallellt ⇒ dela upp beräkningarna
- Uppdelningen definierar uppgifterna
- Uppdelningen definierar DAG-beroendegrafen (riktad, acyklisk, noder=tasks, kanter=beroenden)
- Första (och viktigaste) steget i designen av en parallell algoritm
- Valet av dekomposition bestämmer vilka val som finns senare

Exempel på beroendegraf



Kornighet och parallellitet

- Antalet tasks från dekompositionen och deras storlek bestämmer kornigheten
  - Finkornig vs grovkornig
- Grad av parallellitet – ”level of concurrency”
  - Maximum: största antalet tasks som kan exekveras samtidigt
  - Genomsnittlig: genomsnittliga antalet tasks som kan exekveras samtidigt i hela exekveringen
  - Maximum och genomsnittlig grad av parallellitet beror av kornigheten (ökar normalt med kornigheten)

Kornighet och parallellitet

- P = ”Den kritiska vägen” för en beroendegraf – längsta vägen mellan något par av start- och målnod.
- L = ”Längden av den kritiska vägen”, summan av allt arbete längs kritiska vägen
- W = Totala arbetet i parallella algoritmen
- A = W / L, ger genomsnittliga graden av parallellitet
- => *Kort kritisk väg ger högre grad av parallellitet*
- Kornigheten kan inte/bör inte ökas hur mycket som helst
  - Problemet kan ha inbyggda gränser för kornigheten
  - För hög finkornighet kan ge andra prestandaproblem
    - Dåligt cache-utnyttjande
    - För mycket interaktion mellan uppgifterna

Interaktion mellan uppgifter

- Normalt förekommer någon form av interaktion mellan deluppgifter i alla parallella program
- Kan även ske mellan uppgifter som verkar helt oberoende i programmets beroendegraf
- Interaktionsmönstret kan beskrivas i en uppgift-interaktions-graf (noder=tasks, kanter=interaktion)

Processer och mappning

- Dekompositonen ger deluppgifter som skall exekveras på fysiska processorer
- Mappning är den mekanism som tilldelar uppgifter till olika processer
- Process = mera abstrakt begrepp på enhet som exekverar kod och använder data hörande till speciell task inom ramen för den parallella exekveringen
- Tillåter hierarkisk mappning av tasks inom flera olika programmeringsparadigmer samtidigt
  - Ex: Message passing mellan noder i parallelldator, där varje nod är en gemensamt-minne maskin med >1 CPU:er (t ex på sarek där varje node är en 2-CPU:ers NUMA)

Processer och mappning

- Normalt är #processer = #processorer
- En bra mappning
  - Maximera graden av parallellitet
  - Minimera totala tiden för beräkningsarbetet i det parallella programmet
  - Minimera interaktion mellan processer i den parallella exekveringen
- En bra mappning uppfyller oftast inte alla dessa krav p g a konflikter, t ex mellan parallellitet och interaktion

Rekursiv dekomposition

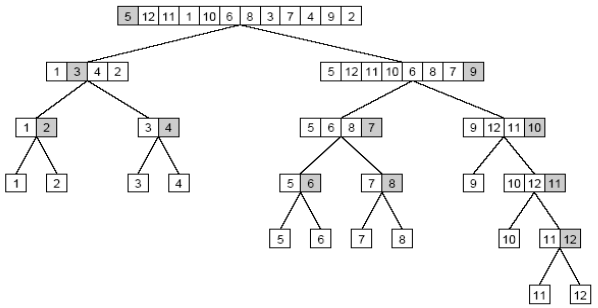
- Att lösa problemet innebär att problemet delas i flera, mindre, problem av samma sort (*divide*)
- Lösningen till de mindre problemen måste kombineras för att få lösningen till det stora problemet (*conquer*)
- Gör många algoritmer enkla att uttrycka
- Varning! Om conquer-steget är stort, kan overheaden bli stor
- Exempel på rekursion: beräkning av explicita matrisinverser av övertriangulära matriser

$$A^{-1} = \begin{bmatrix} A_{11}^{-1} & -A_{11}^{-1}A_{12}A_{22}^{-1} \\ 0 & A_{22}^{-1} \end{bmatrix}$$

- Beräkningsintensivt conquersteg, dock bibehållen komplexitet
- Lars Karlssons exjobb: http://www.cs.umu.se/~larsk/

Rekursiv dekomposition

- Ytterligare exempel på rekursiv dekomposition - med litet conquer-steg: quicksort



Datadekomposition

I de allra flesta parallella algoritmer är det den stora datamängden som är det signifikanta. Två huvudsteg:

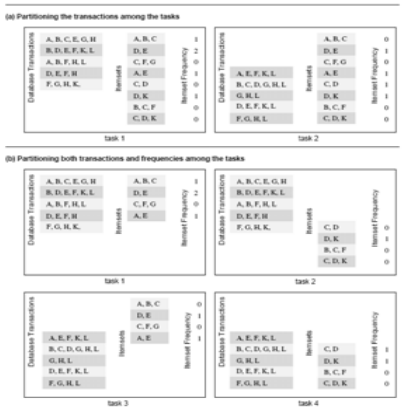
- Datat som beräkningarna skall utföras på partitioneras
- Datapartitioneringen definierar en partition av beräkningsarbetet

Olika varianter:

- Partionera utdata
  - Oberoende utdata? Ex: matrismultiplikation C = A*B
- Partionera indata
  - Hur beräkna utdatat m h a delresultat från partionerat indata?
- Partionera på både in- och utdata
- Partionera på delresultat
  - Uppdelning av mellanliggande delberäkningar

Datadekomposition

Exempel på partition av indata/indata och utdata: beräkning av frekvenser av grupper av transaktioner i transaktions-databas

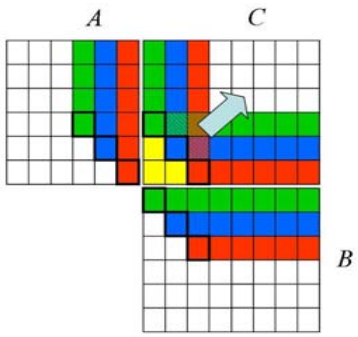


”Ägaren beräknar”-regeln

- Varje datadekomposition av indata och/eller utdata kallas också för ”ägaren beräknar”-regeln.
- Iden är att den som håller en viss del av datat också ansvarar för beräkningarna som hör till den delen av datat
- Exempel: vågfronts-algoritmer för

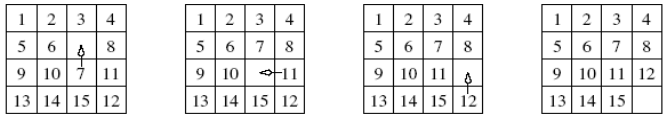
$$AX - XB = C$$

  - Ägaren av C_{ij} ansvarar för beräkningen av X_{ij}
  - C_{ij} skrivs över med X_{ij}



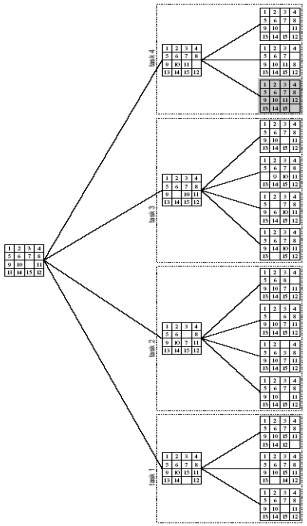
Undersökande dekomposition

- Används vid dekomposition av problem vars lösning beräknas genom sökning i en lösningsrymd
- Grafproblem, spelsökning
- Dela upp problemområdet i delar vars alla resultat inte ”behövs”. Sökningen kan termineras när någon hittat en (tillräckligt bra) lösning



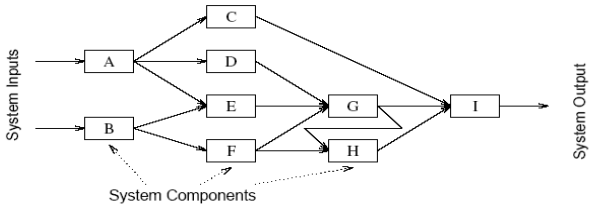
Undersökande dekomposition

- Osäkert hur mycket bättre en parallell algoritm blir: för mycket jobb kan utföras i onödan i jämförelse med seriella varianter
- Speed-up närmar sig p endast om man tar medelvärdet av alla testfall.



Spekulativ dekomposition

- Börja utföra beräkningar trots att alla indata inte är kända. Ex: börja evaluera alla alternativ i en branch (if, switch) innan villkoret för vägvalet är färdigberäknat
- Endast lämpligt när indata kan anta några få värden.
- Garanterat overhead.



Spekulativ dekomposition

- Andra varianter: evaluera bara de alternativ i branchen som verkar mest troliga
- Speedup kan bli signifikant om det finns flera nivåer av spekulativ dekomposition
- Dock $< p$ eftersom det alltid utförs onödigt arbete
- Skillnaden mellan undersökande och spekulativ dekomposition:
  - I undersökande är utdata från multipla uppgifter från en branch okänd
  - I spekulativ är indata till en branch som leder till multipla uppgifter okänd

Hybrid-dekompositioner

- Olika dekompositionstekniker kan kombineras
- Exempel: dekomposition av matrisberäkning på en parallellmaskin med SMP-noder
  - Datadekomposition av input och/eller output mellan noderna
  - Rekursiv dekomposition av arbetet mellan processorerna inom respektive SMP-nod

Uppgifter

- Dekompositionen leder till att olika oberoende uppgifter (tasks) kan identifieras
  - Oberoendet kan ändå innebära viss interaktion
- Uppgifterna skall nu tilldelas de olika processerna
- Hur skapas uppgifter? Statiskt eller dynamiskt?
  - Statiskt: alla uppgifter kända innan exekveringen startar
  - Dynamiskt: alla uppgifter inte kända innan exekveringen startar
  - Dynamiskt skapade uppgifter kräver mer omsorg om lastbalansen och skapar interaktion mellan processer
  - Tilldelning görs vanligen statiskt för statiskt genererade uppgifter och dynamiskt för dynamiskt genererade uppgifter

Uppgifter

- (Beräknings-) storlek på uppgifter?
  - Mängden arbete som krävs för att slutföra uppgiften
  - Uniform/icke uniform – kan påverka lastbalans
- Vet vi storleken på uppgifter?
  - Kan användas vid mappning på processerna
- Storlek på datat som hör till uppgiften?

Interaktion mellan uppgifter

Ex: Kommunikation eller hantering av gemensamt minne

- Statisk – interaktioner och tidpunkter kända a priori
- Dynamisk – interaktioner och tidpunkter icke kända a priori
- Reguljär – interaktioner följer ett givet mönster
- Irreguljär – inget givet mönster

Interaktion mellan uppgifter

- Enbart läsning av gemensamt data (read-only)
- Läsning och skrivning av gemensamt data (read-write)
- En-vägs interaktion initieras och slutförs av en task utan att någon annan blir involverad eller avbruten
  - Kan hanteras av ”gemensamt minne”-paradigmer
- Två-vägs – ”producent och konsument”-scenario
  - Givna modellen för ”distribuerat minne”-paradigmer.
  - Förekommer också i ”gemensamt minne”.

Mappningstekniker för lastbalansering

- Givet en mängd uppgifter, hur placerar vi dessa på processer för att minimera overhead?
  - Minimera kommunikation (interaktion, synkronisering)
  - Minimera väntetid (”idle”)
  - Dessa två mål är ofta i konflikt – finn acceptabel kompromiss
- Förenklat: mappningstekniker statiska eller dynamiska

Statisk mappning

- Statisk mappning distribuerar uppgifterna mellan processerna innan exekveringen
- Statisk mappning ofta kombinerat med datadekomposition
  - Block distribution, högre dimension ger generellt högre parallellitet
    - En-dimensionell. Ex: kolumnblocksmappning av 2-dim array
    - Fler-dimensionell. Ex: mappning av 2-dim array i både rad och kolumnblock
  - Block-cyklisk distribution, många fler block än processorer
    - Bra lastbalans
    - Lämplig då olika delar av datat genererar olika mycket arbete, Ex: LU
    - 2D används i ScaLAPACK
  - Cyklisk distribution
    - Rad eller kolumnvis
    - Perfekt lastbalans men bristande lokalitet kan ge sämre prestanda

Statisk mappning

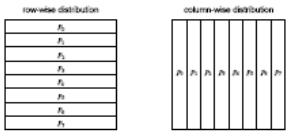


Figure 3.24 Examples of one-dimensional partitioning of an array among eight processes.

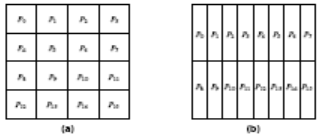


Figure 3.25 Examples of two-dimensional distributions of an array (a) on a 4×4 process grid, and (b) on a 2×8 process grid.

Statisk mappning

- Statisk mappning kan också kombineras med slumpmässig blockdistribution
  - Många fler block än processer
  - Blocken delas ut slumpmässigt
  - Kan vara bättre än t ex block-cyklisk vid glesa matriser

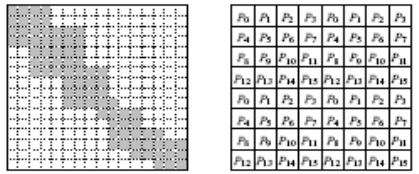
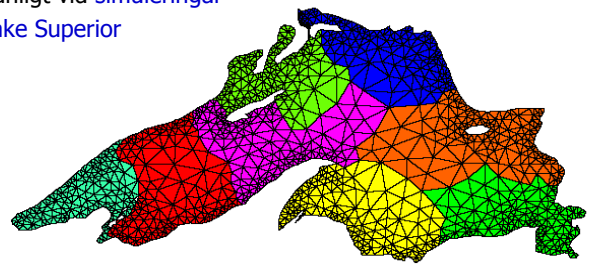


Figure 3.31 Using the block-cyclic distribution shown in (b) to distribute the computations performed in array (a) will lead to load imbalances.

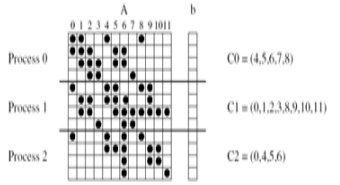
Statisk beroendegraf-partitionering

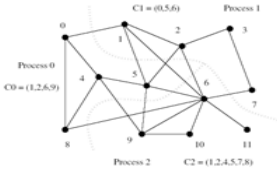
- Dela in datat i delar så att kontaktytorna (=kommunikation) blir så liten som möjligt
- Kontaktytorna bestäms till exempel av en gles matris
- Vanligt vid simuleringar
- Lake Superior



Statisk uppgiftspartitionering

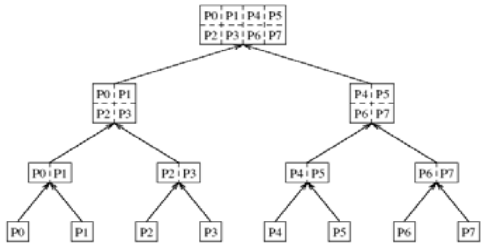
- Dela in uppgiftsgrafen i delar så att kontaktytorna (=kommunikation) blir så liten som möjligt
- Ex: gles matris-vektor multiplikation





Statisk hierarkisk mappning

- Exempel: använd rekursiv mappning för att bygga uppgiftsgraf, och datapartionering för att ytterligare partionera de ”feta” noderna.



Dynamisk mappning

- Nödvändigt då statisk mappning ger obalanser i arbetsbördan mellan olika processer
- Annat namn: dynamisk lastbalansering
  - Centraliserad
    - Master-Slave, centraliserad arbetspool
    - Skalar inte bra: masterprocessen blir flaskhals
    - ”Chunk scheduling” kan lätta på trycket
  - Distribuerad
    - Vilka par av processer skall utbyta arbete?
    - Vem tar initiativet? Sändaren eller mottagaren?
    - Hur mycket arbete skickas i varje sändning?
    - När skall arbete utbytas? När det är slut? När man vill bli av med lite?
- Kan implementeras i de flesta paradigmer

Metoder för att dölja interaktion

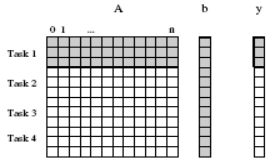
- Maximera datalokalitet
  - Minimera totala mängden data som utbytes
    - Välj lämplig dekompostion och mappning
    - Ex: distribution av högre dimension oftast bättre
  - Minimera frekvensen av datautbyte
    - Omstrukturering av parallella algoritmen kan vara nödvändig
    - Kommunicera större delar data vid färre tillfällen
- Minimera flaskhalsar
  - Trängsel (contention)
  - Kommunikation
  - Omstrukturering av parallella algoritmen så att beräkningar utföres på klart avgränsade delar

Metoder för att dölja interaktion

- Överlappa beräkningar med kommunikation
  - Mycket vanligt vid numeriska beräkningar
  - Nackdel: minskar kornigheten på uppgifter
  - Kräver stöd av underliggande mjukvara och/eller hårdvara
    - Distribuerat minne – räkna och kommunicera samtidigt
    - Gemensamt minne – data prefetching
- Använd kollektiva kommunikationsoperationer
  - Broadcast, scatter, reduce, gather, all-to-all
  - Ofta högoptimerade implementationer, t ex *recursive doubling*
- Överlappa interaktioner med andra interaktioner

Metoder för att dölja interaktion

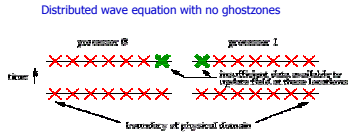
- Replikera data
  - Kräver extra minne.
  - Ex: matris-vektor y = Ab, alla processer har hela vektorn b

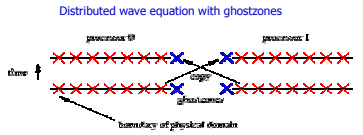


- Utför extra beräkningar (redundant computing)
  - Ex: spökceller i beräkningsnät för tidsberoende PDE,
  - Behöver bara kommunicera varannat tidssteg

Metoder för att dölja interaktion, exempel

- För att uppdatera värden på inre gränser behövs värden från andra processorer
- Man kan utbyta O(n) datapunkter i varje tidssteg...
- Man kan också välja att utbyta O(2n) datapunkter i vart annat tidssteg och beräkna de O(n) värden som behövs för nästa steg på båda sidor om gränsen
- Extra datat sparas i ”spökzoner” Sparar kommunikation – gör dock extra arbete (ofta försumbart vid grovkornig dekomposition)





Modeller för parallella algoritmer

- Data-parallell modell
  - Identiska operationer utföres parallellt på stora datamängder
  - Data-partitionering, statisk mappning, reguljära interaktioner
  - Ex: applicera en beräkningsstencil för PDE över ett diskretiserat område
  - Stora problem kan lösas effektivt
- Uppgifts-parallellism-modell
  - Partitionering av beroendegraf
  - Statisk/dynamisk partitionering och mappning
  - Ex: kompilera oberoende subrutiner, anropa oberoende subrutiner
- Arbetspoolsmodell
  - Dynamisk mappning för god lastbalans
  - Centraliserad eller decentraliserad (se tidigare)
  - Ex: lösa spel- eller grafproblem genom sökning i ett lösningsrum

Modeller för parallella algoritmer

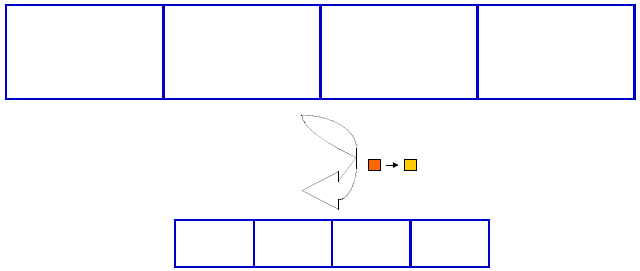
- Master-slave
  - En process genererar arbete och distribuerar till arbetare
  - Uppgifterna kan genereras och fördelas statiskt på förhand om möjligt
  - Mastern kan bli flaskhals
  - Ex: distribution av oberoende iterationer i en loop på SMP-maskin
- Pipeline
  - Ström av data passerar genom olika processer
  - Olika processer gör olika saker med datat parallellt
  - Kedja av producent-konsument
  - Ex: Parallell LU-faktorisering
  - Se även *fallstudien*
- Hybridmodeller
  - Olika modeller appliceras hierarkiskt eller sekventiellt på ett problem

Fallstudie: MPEG-2 till MPEG-1 transcoding

Följande ska utföras:

- En MPEG-2-ström ska omvandlas till en MPEG-1-ström
- Viss korrigering av färgerna ska ske
- MPEG-1-upplösningen är *halva* MPEG-2-upplösningen
  - Från full TV-upplösning till ”skräp-video”
- Både in och ut-format är ”long-GOP” (group of pictures)
  - Förenklat:
  - Var 12:e ruta är en referensruta (I)
  - Resten byggs upp med differens från föregående I-frame (P-frames, betydligt mindre än I-frames, P = prediction)
  - I verkliga livet finns också B-frames: differenser som pekar bakåt till närmaste I eller P eller framåt mot närmaste I eller P, B = bi-directional. B-frames möjliggör mera komprimering.
- Typisk lagring av strömmen: IBBPBBPBBPBBIBBPBBPBBPBBI osv...

Fallstudie: MPEG-2 till MPEG-1 transcoding



Fallstudie: MPEG-2 till MPEG-1 transcoding

Dekomposition:

- GOP
- Ruta
- Block
- Hierarkisk
- Uppgift
- ...?

Fallstudie: MPEG-2 till MPEG-1 transcoding

- Förslag till lösning:
  - Pipeline: alla bilderna strömmar igenom pipelinen där olika processer ansvarar för olika steg
    - Uppackning – bygg bilden av resp I-,P- och B-frames
    - Omskalning
    - Färgkorrigering
    - +några andra operationer (ex. konvertera/mixa ljud)
    - Nedpackning – ta isär bilden till resp I-,P- och B-frames
  - Eventuellt går flera processer in i ett av stegen om det är för kostnadskrävande i jämförelse med de andra stegen
    - Rutorna delas upp blockvis i rektanglar (1-dim datapartitionering)
  - Task-parallellism
  - Lämplig för SMP eller multi(dual/quad)-core maskiner
    - Stöd för både tunga processer (fork) och lättviktstrådar (threads)

Sammanfattning – check box

Dekomposition	Rekursion	Data	Undersökande	Spekulativ
Uppgifter	Hur skapas	Storlek	Kunskap om storlek	Storlek på data
Interaktion m. uppgifter	Statisk/ dynamisk	Reguljär/ irreguljär	Läsning/ skrivning	Sändning/ utbyte
Mappning	Statisk:	Array/block/ cyklisk/slump	Graf	Uppgift/ hierarkisk
	Dynamisk:	Centraliserad	Decentraliserad	
Hantera overhead	Maximera datalokalitet	Minimera flaskhalsar	Replikera data/beräkn.	Överlappa/ gruppkommun.
Modell	Dataparallell	Uppgiftsgraf	Arbetspool/ master-slave	Pipeline