**3. Паралелно програмиране. Принципи на разделяне и балансиране на програмите. Синхронни и асинхронни паралелни приложения. Параметри, метрика, анализ. Системни средства за паралелно програмиране.**

**Последователни и паралелни програми**

Програмата се състои от процеси, които могат да бъдат изпълнявани последователно или кон-курентно. 1) При изпълнение на програма в ***среда за последователното програмиране***: -про-грамата се състои от един процес; -резултатът от изпълнението й с еднакви данни винаги е един и същ; -изпълнението на всяка инструкция е последователно и независимо от изпълне-нието на др. инструкции. 2)При изпълнение на програмите в ***среди с мултипрограмитене***: -програмата се състои от един птоцес; -управлението се предава последователно м/у различни процеси; -м/у отделните процеси съществува зависимост по време на изпълнение, но резулта-та от изпълнението им се запазва. 3)При изпълненеи на програмите в ***среди за паралелно про-грамиране***: -програмата се състои от множество паралелни (конкуриращи се) процеси; - тя включва освен управляващ код и данни, също и инструкции за синхронизация и обмен м/у про-цесите, които съставляват нейния планиращ процес (scheduler); -резултатът от изпълнението на паралелната програма може да зависи от работата на планиращия процес.

**Паралелни процеси**

Процесите, изпълняващи програмата за паралелна обработка, могат да бъдат алтернативно: 1)***Реплики***, изпълняващи еднакви подпрограми в/у различни данни – модел SPMD (Single Program Multiple Data).Разликата от SIMD е, че в този случай синхронизацията се извършва на ниво подпрограма, а не на ниво инструкция и затова SPMD обслужване се изпълнява на MIMD компютри; 2) ***Различни подпрограми*** – модел MPMD (Multiple Program MultipleData). При този подход отделните подпрограми-процеси се пораждат като дъщерни на един (главен) процес.

**Граф на процесите ([precedence | dependency] graph)**

1)Зависимостта по данни и управление се изследва (чрез графи) на различни нива – блок,израз, променлива; 2) Компилаторите обикновено изследват графа на зависимостие на ниво израз и променлива – пример за серията изрази: S1: A=B+C, S2: B=A+E, S3: A=A+B. S1, S2, S3 – statements – представяме си ги като изпълнявани от различни процеси, намиращи се в един и същ контекст; те ще се конкурират за достъп до едни и същи променливи: A, B. Изразите се изобразяват като възли в графа на зависимостите, а дъгите са зависимостите като началото на дъга е променлива (аргумент или стойност) на израз, а край – същата променлива от следващ израз – освен когато началото и края на дъгата са аргументи (от дясната страна) на изразите. (**фиг.3.1**)

**Типове зависимости в графа на процесите: (фиг.3.2)**

1) ***Зависимост по данни (data flow)***: резултата от израз е аргумент на следващ израз (прена-реждането на изразите или паралелното им изпълнение променя резултата на следващия из-раз ) – тази зависимост е непреодолима; 2) ***Антизависимост (anti-dependency)***: аргумента на израз е резултат от следващ израз (пренареждането на изразите или паралелното им изпълне-ние променя резултата на анализирания израз) – тази зависимост може да бъде преодоляна чрез репликиране на променливите; 3) ***Зависимост по изход (data output)*** – резултатите от два израза се записват в една и съща променлива (пренареждане или паралелно изпълнение про-меня стойността на тази променлива)– тази зависимост може да бъде преодоляна чрез репли-киране на променливите; 4) ***Зависимост по вход (data input)***: два израза имат общ аргумент – тази зависимост няма значение при съвременните програмни системи (поради средствата за конкурентен достъп); 5) ***Зависимост по управление (data control)***: условно изпълнение на из-раз, където условието е резултат от предходен израз (разновидност на зависимостта по дан-ни).N.B.: За по-висок паралелизъм на кода се отстраняват антизависимостите и зависимостите по изход.

**Пример за отстраняване на зависимости**

|  |  |
| --- | --- |
| Изходен код | Код с намалена завиc. |
| for i=1, n, 1  **x**=A[i]+B[i]  Y[i]=2\***x**  **x**=C[i]\*D[i]  P=**x**+15  endfor | for i=1, n, 1  **x**=A [i]+B[i]  Y[i]=2\***x**  **xx**=C[i]\*D[i]  P=**xx**+15  endfor |

**Модели обща памет**

***1)******В******паралелните системи*** достъпът до общата памет и ресурси за В/И е конкурентен и се ба-зира на схемите за PRAM (Parallel Random Access Machine) – автономни процесори с конкурен-тен достъп до обща памет (която включва и В/И канали); ***2)в модела*** ***PRAM*** се предлагат 4 схе-ми за отстраняване на конфликтен конкурентен достъп до общото адресно пространство: **-ЕREW** **(Exclusive Read, Exclusive Write)** – резервиране на конкурентния достъп да даден адрес за двата типа операции; -**CREW (Concurrent Read, Exclusive Write)** – няколко про-цесора могат да четата едновремнно даден адрес, но операциите за запис са монополни; -**ERCW (Exclusive Read, Concurrent Write)** – допускат се няколко едновременни операции на запис но монополно четене; -**CRCW (Concurrent Read, Concurrent Write)** – конкурентните опера-ции са без ограниче-ние; ***3)\*\*EW*** схемите съответстват на изискванията за консистентност (съгласуваност и детер-министичност) на данните и се прилагат като универсални при повечето паралелни алгоритми; ***4)***Конкурентните операции за запис при ***\*\*CW схемите*** имат ограничено приложение при ня-кои класове паралелни алгоритмиза обработка на графи и числова обра-ботка, при които по-стигат по-високо бързодействие от схемите с резервиран запис.

**Модел с обмен на съобщения**

При обмен на съобщения всяка двоика процеси е свързана с комуникационен канал, редставен с точно една променлива – последователните съобщения са стойностите на тази променлива; дефинирано е състояние на канала – напр. четене на променливата-канал се допуска само ко-гато състоянието му не е празен (респ. при запис – да не е пълен); синхронният и синхронният канал са с еднакъв режим на достъп но асинхронният има капацитет = размера на буфера (>1).

**Паралелни алгоритми**

***1)*** Паралелните алгоритми са междинното звено във веригата на паралелната обработка (меж-ду изчислителния проблем и паралелната система) –архитектура, система/среда, програма, ал-горитъм, изчислителен проблем; ***2)*** Паралелния алгоритъм е абстрактно (формално или нефор-мално) представяне на изчислителен проблем като набор от процеси за едновременно изпъл-нение; ***3)*** Основните характеристики на паралелния алгоритъм (които отсъстват при посл. алго-ритми) са: ***брой процеси и логическата им организация*** (***master-slave***: примерно клиент-сър-вър – двата са инициатините процеси = master; slave- дефинират изпълнител-ната част на зада-нието), ***разпределение на данните*** (***декомпозиция +*** възможности за ***разпределена алокация***: конвейеризация), *точки на синхронизация* (оптимизиране), *модел на междупроцесния обмен* (основно обща памет – обмен на съобщения);***4)*** Разл. конкретни реше-ния на горните характе-ристики пораждат цял клас от ПА, базирани на един последователен алгоритъм.

**Фази на проектирането на паралелен алгоритъм**

Проектир. на парал. алгоритъм минава през следните фази: ***1) Разделяне*** (***partitioning***) – декомпозиция на проблема: **по данни** (главно **SPMD**) или **по функции** (главно **MPMD**) – разде-лянето се извършва с оглед на спецификата на проблема; целта е да се дефинират множество подзадания; грануларността при тази фаза не отчита особеностите на архитектурата, която ще се използва за обработка – резултатът от фазата е дефиниция на отделните задания; ***2) Кому-никации (и зависимост)*** (***communication***) – формулира информационните или контролните за-висимости между отделните подзадания; комуникациите се представят като канали и съобще-ния, които се предават по тези канали; архитектурата за обработка се игнорира и на тази фаза, но специфицирането на каналите помага да се оцени алгоритъма по комуникационна слож-ност; ***3) Формиране*** (***agglomeration***) – след оценка на изчислителната и комуникационната сложност на формулираните подзадания и прилежащите им комуникации, те се групират в за-дания, при което се отчитат характеристиките на архитектурата на обработка – основно

брой процесори/възли и комуникационен модел – и в резултат се постига оптимизиране по следните характеристики: грануларност и балансираност (с оцека на изчислителната сложност на отделните задания), евентуално репликиране на данни и подзадания, оптимизиране на ко-муникациите (с оцека на комуникационната сложност на отделните задания), евентуално за-пазване на линейност (скалируемаст), технологично оптимизиране (напр. намаляване на раз-ходите за кодиране на заданията); ***4)Разпределяне*** (***mapping***) – незадължителна фаза (отсъст-ва при проектиране на паралелен алгоритъм за системи с динамично планиране- обикн. мулти-процесор с разпределена ОС), която се със-тои в разпределяне на формираните задания (или евентуално групи от задания) по обработва-щите възли на системата със кодиране на съответното решение. N.B.: обикновено се използва специален език за спецификация на зареждането и евентуално за настрока на комуникацион-ните канали напр. в системи с комутируеми канали, така че от алгоритъма се изисква да специ-фицира и комуникациония граф на системата за обработка(**фиг. 3.3**).

**Метрика и анализ на производителноста**

***1) Сложността на последователните алгоритми*** (брой операции) се оценява като функция само на размера на проблемната област и следователно може да се оцени абстрактно от архи-тектура-та; при паралелните алгоритми тя е функция на архитектурата и на средата за паралел-на обработка (особено при динамично планиране); ***2)***Основен фактор при паралелните алго-ритми е ***степента на паралелизъм Р*** – макс. бр. операции, които могат да се изпълнят пара-лелно при обработката на алгоритъма – това е архитектурно-независима величина; при размер на проблема **W** не повече от **P(W)** процесора могат де се ползват ефективно; съществено е съотношението между паралелните и последователните сегменти на паралелни алгоритми.

**Закон на Amdahl (1967):**

При наличие на две интензивности (R – rate = интензивност) на обработка на даден порблем – ***високо-паралелна Rh (rate-high)*** и ***ниско-паралелна Rl (rate-low)***, които са в съотношение *f*:(1-*f*) по брой на генерирани резултати (междин-ни и крайни) – общата интензивност на обработка е *R*(*f*) = [*f*/*Rh* + 1-*f*)/*Rl*]-1, следователно *f* → 1 *R*(*f*) → *Rh* и при *f* → 0 *R*(*f*) → *Rl. (*N.B.: макар че е формулиран за темпове на обработка, закона е в сила и се прилага за агрегирана степен на паралелизма на заданието). (**фиг.3.4**)

**Ускорение и ефективност**

***1)*** При ***оценка*** ***или измерване на ускорението*** (***Sp* = *T*1/*Tp***) се приема, че всички процесори в двата случая са с идентична производителност; поради наличие на комуникационни и синхро-низационни закъснения **1 < *Sp* < *p; 2)******Аномалии***: -***суперлинейно Sp* > *p***може да се наблюдава при неоптимален последователен алгоритъм или особени характеристики на проблема, които изявяват нисък капацитет на използвания хардуер: напр. при голям размер на данните е въз-можно значително закъснение на последователната обработка на проблема поради бавни операции с външната памет, докато при паралелна обработка с разделянето на данните между възлите този проблем отпада ; - ***немонтонно Sp1* > *Sp2***за ***p2* > *p1***– често срещана аномалия

***3) Ефективността***, която е нормализирано ускорение (***Ep* = *Sp*/*p* = *T*1/(*pTp*) < 100%**), характе-ризира частта от общото време за паралелна обработка,през която процесорните елементи се използват.

**Пример за оценка на ускорението и ефективността**

Хиперкуб от *p* процесора изчислява сумата на *n* числа. Времето за локална операция „+” на две числа и времето за предаване резултата на съседен процесор е 1. Паралелният алгоритъм: чи-слата се сумират локално за време *n*/*p,* след което локалните (първоначално *р* на брой) парци-ални суми се предават на съсед (1) и сумират (1) за 2lb*p* (**фиг.3.5**). *Sp* = n/[*n*/*p +* 2lb*p*] = *np*/[*n* + 2*p*lb*p*]; *Ep* = *n*/[*n* + 2*p*lb*p*]. Получените зависимости показват обичайният ефект при по-голямата част от ПА на намаляване на ефективността с нарастване на *р* (при фиксиран размер на проблема *n*) – ефект от закона на Amdahl.

|  |
| --- |
| **Фиг. 3.5** Описание от лекции |
| (1)  (2) Преносът от долните към горните – единица време за операция.  (3) Преносът от горните-предни към горните задни.  (4) Преносът към едно от задните. |

**Цена и коефициент на използване**

***1) Цена (cost)*** при обработката на парал. алгоритъм с **p** процесора за **Tp** единици време (N.B. единица време е времето за изпълнение на една елементарна операция) е ***Cp* = *pTp ,*** т.е. **Cp** е макс. бр. операции, които биха могли да се извършат за времето на обработка на съответния парал. алгоритъм; ***2) Коеф. на използване (utilization)*** при обработката на парал. алгоритъм, състоящ се от **Ор** действителния бр. операции с **p** процесора е ***Up* = *Op*/*Cp* = *Op*/(*pTp*)**, т.е. **Up** e отношението на действителните към потенциалните операции при обработка на съответния парал. алгоритъм.

**Темп и излишък**

***1) Темпът на обработка (execution rate)*** e архитектурно-зависим параметър и се представя с няколко скали: ***MIPS*** (унипроцесори, мултипроцесори), ***MFLOPS*** (SIMD, числова обработка), ***MOPS*** (SIMD), ***LIPS [# logic inferences p.s.]*** (**AI** приложения, където AI – изкуствен интелект). Освен по архитектурен критерий, изборът на скала зависи и от типа парал. алгоритъм, които се обработват; ***2) Излишък*** ***(redundancy)*** при обработката на парал. алгоритъм, състоящ се от ***Ор*** на брой операции при обработка с **p** процесора е **Rp = Op/ O1** > 1 (където *О*1 е броя операции при обработка на уникомпютър), т.е. **Rp** e критерий за свръхтовара, който се поражда от паралеланата обработка на алгоритъма; **p** и **n** (размера на проблема) са аргументи на **Rp,** но в зависимост от класа парал. алгоритъм участват с различна тежест. Освен ***redundancy*** има и (System) ***overload.***

**Алгоритмична сложност**

Коректността на даден парал. алгоритъм е архитектурно-независима, но неговата ефективност зависи от изпълнителната платформа, поради което е целесъобразно сложността му да се оценява и като функция на разпределянето (mapping). По принцип **алгоритмичната сложност *О*** оценява времевите и пространствени характеристики на обработка – **времевата сложност *Т***се задава в брой елементарни операции и комуникации (от който се получава времето за обработка в дадена архитектура), а **пространствената сложност *М***в брой алоцирани регистри и клетки памет (т.е. *О* = *О*(*Т*, *М*)); Оценкта се дава обикновено като долна и горна граница на тези величини или с приближение – асимптотична сложност.

**Паралелно програмиране в разпределени системи**

Прилага моделите: 1) Разпределена обща памет (**DSM =Distributed Shared Memory**): ключалки семафори, монитори, бариери; 2) Обмен на съобщения (**Message Passing Systems**): - *приложно-ориентиран* *междинен слой* (**MPI** и **PVM** – процедурен модел; **RMI и Corba** – обектен модел); -йерархични (**master-slave**, **client-service** - Jini) и нейерархични модели (**P2P - Jxta**).

**Конвенционален псевдокод за паралелни алгоритми**

|  |  |
| --- | --- |
| Псевдокодът e приложим за определени класове архитектури – обикновено се взима като предпоставка най-разпространения PRAM модел за паралелен достъп до обща памет – CREW декларация на процедури и функции е разширена със запис на модела за паралелна обработка и броя алоцирани процесори: | **Procedure: <name> ({list of parameters})**  Model: <model name> with p = f(n) processors  **Input: <input variables>**  **Output: <output variables>**  **Declare: <[global and] local variables>**  **Function: <name> ({list of parameters})**  Model: <model name> with p = f(n) processors  **Input: <input variables>**  **Output: <output variables>** |

**Блок FORALL**

Този блок се прилага за имитация на паралелно изпълнение на вложения в негосегмент (набор изрази) – асинхронно (в MIMD) или синхронно (в SIMD) синтаксис:

|  |  |
| --- | --- |
| FORALL identifier: RangeType IN {PARALLEL | SYNC}  Statement\_1  …  Statement\_K  END | **identifier** е упр. пром., деф. в границите на блока; по 1 процес се създава за вс. нейна ст-ст (м-вото ст-сти трябва да е крайно); в създ. процеси identifier има разл. ст-сти; **RangeType** е типът на упр. пром., чиято мощност освен това задава и бр. парал. процеси; **PARALLEL** или **SYNC** задава типа парал. обработка – съотв. асинхронен илисинхронен |

Изпълнението на блока завършва след изпълнение на всеки от процесите. Асинхронната обработка означава, че част от процесите могат да се планират след изпълнението на другите (когато броят им е по-голям от броя процесори).

**Пример за блок FORALL**

|  |  |
| --- | --- |
| 8 процеса за асинхронна паралелна бработка на функция с аргумент – номера на процеса | Версия |
| **FORALL** x:[1..8] IN PARALLEL  y = some\_function(x);  **END** | FORALL x ∈ X IN PARALLEL do y = some\_function(x); |

**Израз do IN PARALLEL**

Този израз се прилага като директива в различни блокове.

|  |  |
| --- | --- |
| Пр.: при парал. векторна бработка интаксис: | Пр.: за вс. елем. на масивите се формира отделен процес |
| for <израз в/у инд. на масив> do IN PARALLEL  Statement\_1  Statement\_2  …  Statement\_K  end IN PARALLEL | for i = 1 to n do IN PARALLEL  read(A[i], B[i])  if (A[i] > B[i])  then write(A[i])  else write(B[i])  endif  end IN PARALLEL |

**Синхронизационни конвенции, семафори**

***1) Синхронизационните схеми*** биват: контрол на достъп (семафори и монитори) и контрол за последователност (бариери). ***2)*** Променлива от тип ***семафор*** се асоциира с всеки адрес за общ достъп и върху нeя се извършват операциите: *установяване на състоянието* (активно или пасивно) (***wait***), *блокиране на процес* (***wait***), *възстановяване от блокиране* (**signal**). ***3) Wait(S)*** e заявка за достъп до критичната зона, която се потвърждава ако S>0 (и S се декрементира); в противeн случай процесe блокира и изчаква. ***4)*** Signal***(S)*** освобождава критичната зона, инкрементира **S** и възстановява чакащ процес.

|  |  |
| --- | --- |
| **Синхронизиращ псевдокод със семафор** | P1: wait(S1)  {critical section 1}  signal{S1}  P1: wait(S1)  {critical section 2}  signal{S1} |

|  |  |
| --- | --- |
| **Синхронизация с монитори**  1) Mониторите са разширение на семафорите, което се състои както от данните за контрол на достъпа – ***condition variable***, така и от процедурите – ***signal*** и ***wait***  2) При дефиниране на ***condition variable*** се създава и опашка на идентификаторите на чакащи процеси, които се възстановяват и получават достъп до критичната зона с операцията ***signal*** | Monitor Resource\_alloc  Var Resource\_in\_use: Boolean;  Resource\_is\_free: Condition;  Procedure Get\_resource  begin  is (Resource\_is\_free) then  wait(Resource\_is\_free)  Resource\_in\_use = true  end  ...  Procedure Release\_resource  begin  Resource\_in\_use = false  signal(Resource\_is\_free)  end  end Monitor |

|  |  |
| --- | --- |
| **Синхронизация с бариери**  1) С бариерите се осъществява контрол за  последователност – напр. за запазване на зависимостта по данни  2) Бариерата също се състои от буфер за готови изчакващи процеси и боряч | ***Псевдокод без синхронизация***  For I = 1 to N do IN PARALLEL  { S1: A[I] = func\_a(A[I])  S2: B[I] = func\_a(B[I])  S3: C[I] = func\_c(A[I], B[I])  } |
| ***Псевдокод с бариерна синхронизация***  For I = 1 to N do IN PARALLEL  { S1: A[I] = func\_a(A[I])  S2: B[I] = func\_a(B[I])  **BARRIER(2)**  S3: C[I] = func\_c(A[I], B[I])  } |

**Задачи на балансирането на изчислителния товар (Load Balancing – LB, Resource Management, Resource/Job Scheduling)**

***1) Минимизиране времето*** за решаване на даден пробл. при парал.обраб. чрез изравняване на локалното натоварване на обраб. възли. ***2) Целта*** може да бъде не пълно изравняване а недопускане на възел в престой, докато трае парал. обраб. ***3) В грид*** – пропорцинално натоварване на ресурси с различна собственост и администрация. ***4)Източници* *на дисбаланс:*** нерегулярност на проблема при паралелизъм по данни; недетерминистични алгоритми за обработка (напр. при неизвестен бр. итерации за достигане до решението – търсене в графи и др.); невъзможно или некомпетентно декомпозиране – при паралелизъм по данни или по управление.

**Статично балансиране**

***1)*** Разпределянето на заданията по възли и алоцирането на ресурси се извършва (и е известно) преди да стартира паралелната обработка – ***планиране, комплементиране*** (**mapping**, **matchmaking**, **scheduling**). ***2)Подходи*** за статично балансиране: **RR** – ***циклично алоциране*** на заданията по обработващи процеси; ***стохастично разпределяне; рекурсивно разделяне*** – при алгоритмите за графи – бисекция (разделяне на проблема на подпроблеми с очаквана еднаква сложност на обработка и с генериране на минимален синхронизационен и комуникационен свръхтовар); ***генетични и Монте Карло алгоритми*** – свързани са с генериране на възможни варианти на декомпозицията и оценяването им, така че да се избере оптималния.

**Недастатъци на статичното балансиране**

***1)*** Проблемна предварителна оценка на сложността на подпроблемите, получени при декомпозицията. ***2)*** Не може да отчете текущото състояние на ресурсите по време на обработката – фоновото натоварване на ресурсите (процесорницикли, памет, комуникационни канали), както и реалните синхронизационни и комуникационни закъснения – ограничено приложение за синхронни алгоритми. ***3)*** При недетерминистични алгоритми за обработка, напр. при неизвестен брой итерации за достигане до решението – търсене в графи и др. – статично решение на задачата за товарен балнс е невъзможно освен чрез прилагане на по-фина грануларност и откриване на край (distributed termination detection).

**Динамично балансиране**

***1) Разпределянето*** на заданията ***по възли и алоцирането на ресурси*** се извършва по време на паралелната обработка и е известно едва след приключването й. ***2) Централизиран подход*** – ***master-slave*** обработка; декомпозицията, разпределянето на заданията и ресурсите, откриването на край или алтернативно интегрирането на резултата са функции на един ***master*** процес. ***3) Разпределен подход*** – декомпозиция на управляващия процес в йерархия от упр. процеси или асоцииране на упр. функции с всеки от обработващите процеси. (**фиг. 3.6**)

|  |
| --- |
| Описание на **фиг. 3.6**, където I- Information, L- Location, T- Transfer |
| (I) Услугата за наблюдение се нарича monitoring – следи натоварването в отделните възли.  (L) Match-ва даден ресурс.  (T) Извършва се трансфер на данните. |

**Централизирано динамично балансиране**

***1)*** ***Главния процес*** функционира като ***пул от задания*** (***work pool***) и получава заявки за ново задание от готовите изпълнителни процеси; изпълнителните процеси са обикновено реплики (модел SPMD). ***2) Пулът от задания*** се прилага при матричните изчисления, при алгоритмите “разделяй и владей”. ***3) Нерегулярните и динамичните товари*** също са подходящи за ***work pool*** обработка – в посл. случай генерираните от обработката нови задания се присъединяват в опашката на пула заедно с тек. резултат от изпълнителния процес (**фиг. 3.7**) ***4)Осн. предимство*** на централизираното динам. балансиране е лесното установяване на изпълнение на условието за край – при празен пул и прекратена работа на изпъл. процеси; при някои алгоритми за търсене условието за край се открива от някой от изпълнителните процеси и се предава към главния процес заедно с резултата. ***5) Недостатък*** е възможността за възникване на тясно място и ниската линейност. (**фиг. 3.9** – BDP = Birth-Death Process)

**Разпределено динамична балансиране**

***1)*** Пряк подход е ***разпределяне на функциите*** на упр. процес по поддържане на динамичния пул от задания на йерархичен слой на упр. процеси – **фиг. 3.8**. ***2) Оптимизацията*** в горния случай е предимно в избора на брой упр. процеси от втори ниво или евентуално избор на броя упр. нива. ***3)*** При някои алгоритми се практикува развито ***йерархично дърво*** – обикновено двоично, тъй като разделянето на [под-]проблема на две очаквано равно части е по-лесно за алгоримиране и за прилагане на рекурсия.

**Р2Р динамично балансиране**

***1)*** То е ***форма*** на по-пълно прилагане ***на разпределеното динамично балансиране***. Премахва се разделението на упр. и изпълнителни процеси като всеки процес извършва и двете функции. **2)** Формално и опростено цялото задание може да бъде предадено за ***изпълнение в един процес/възел***, след което се извършва неговата декомпозиция и послеващ балансиращ трансфер на генерираните подзадания межу възлите. ***3)*** В този случай ***декомпозицията*** е желателно да бъде ***или тривиална*** (примерно при матрични изчисления), ***или*** пък да бъде ***опростена*** (примерно бисекция на проблема без първоначален анализ колко са потенциалните обработващи процеси, какво е тяхното текущо натоварване и каква е

оптималната грануларност).

**Параметри на Р2Р динамичното балансиране**

***1)*** Подобни ***балансиращи схеми*** се наричат ***дифузионни***, тъй като реализират балансирането чрез трансфер на подзадания към “съседни” възли; релацията за съседство в случая може да изхожда от конкр. топология на изпълнителната платформа, но може да бъде и подчинена на разл. стохастични принципи – напр. на случ. избор от опр. бр. (оптимизационен параметър!) “съседи”. ***2)*** В горния случай като средство за повишаване на линейността на алгоритъма се избягват схеми, когато всички възли са “съседни”; вместо това се ***формират виртуални топологични структури*** – линия, пръстен, хиперкуб и др. (обикн. нейерархични) топологии; когато валентността на процесите е по-голяма от 1, може да се прилага циклично или случайно тъсене на “съсед” за балансиращ трансфер. ***3)*** Др. важен парам. на Р2Р балансиране е ***инициативата*** (или момента за активиране на лок. балансираща процедура): инициатива на донора, инициатива на приемника.

**Системи за динамично балансиране**

***1)*** ***Информационна, локационна и трансферна стратегия*** – функции, граф, разпределение; (клъстерно, мултиклъстерно и c2c планиране). ***2) Синхронно балансиране*** – co-scheduling: **Koala. *3) Асинхронно балансиране*** – htc (High troughput computing), volunteer computing:

**Condor/Condor-G , Boinc;** балансират се нископриоритетните процеси на опортюнистичнитепотребители във фонов режим (background priority).

**Еталонни паралелни алгоритми**

***1) Асинхронни*** алгоритми – Mandelbrot set. При **Mandelbrot set:** имаме една функция в/у комплексната равнина. Тя се прилага в/у комплексен елемент и резултатът също е компл. число. Така итерираме и получаваме числа близки илине до (0,0). Изследваме бр. необходими итерации за достигане на (0,0). Ако същ. такъв брой, казваме, че това е число от ***Mandelbrot set***. Това наподобява фрактални изчисления. Ако пуснем теста на ***Mandelbrot*** в/у една технология: имаме матрица, чиито елемент са цветове и оцветяваме в по-светло, ако проблемът се е решил по-бързо. **Фиг. 3.10.А** 🡪тъмното петно значи технологията не е много подходяща. Ако имаме 4 ядра разделяме матрицата на 4 равни части. **Фиг.3.10.B** 🡪 в това ядро (долу в дясно на схемата) тъмното надделява, значи на него се е паднала най-тежката задача. ***2) Локално-синхронни*** алгоритми – Watеr simulation, odd-even sort. При Water simulation: синхронно е, защото стъпка по стъпка се развива симулацията. Можем пак да направим декомпозиция на тероида: гранулността тук не е много ефективна, т.к. всеки процес ще има по 4 съседни и трябва да има комуникация м/у тях – деф. на релация на съседство. Например модела хищник-жертва. **Фиг.3.11** 🡪 при стартиране на симулация обектите преминават в съседно квадратче, ако то е свободно – това важи да жертвите, а хищниците отиват там, където има жертви. ***3) Глобално-синхронни*** алгоритми – n-body simulation, Ray tracing. При ***n-body simulation*** имаме n точки с планарни координати. Точките нямат форма и размери, но имат свойства – напр. маса и ускорение (пулс). **Фиг. 3.12.A** 🡪Каква ще бъде траекторията на всяко едно от тези тела? Функция на взаимодействие м/у двойките тела. Всяко тяло има свой пулс и трябва да знаем масата и текущото положение. Вземаме гравитационната сила и така може, да изчислим новото място. Това правим за всеки. Но някои могат да се сблъскат, т.е. пак е нужна комуникация. Може пак да разпределим м/у p процесора, които ще си общуват чрез shared memory или message passing. **Фиг. 3.12.B** 🡪 динамичен LB – рекурсивно разделя на подобласти.