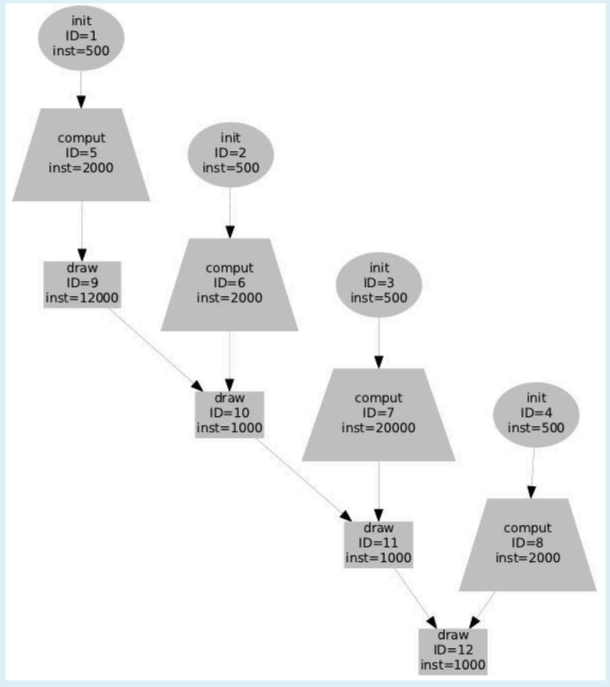
Guió i material per classe no presencial PAR grup 20
Dilluns 22/2/2021, 10:00-12:00

Repàs vídeo lesson 2

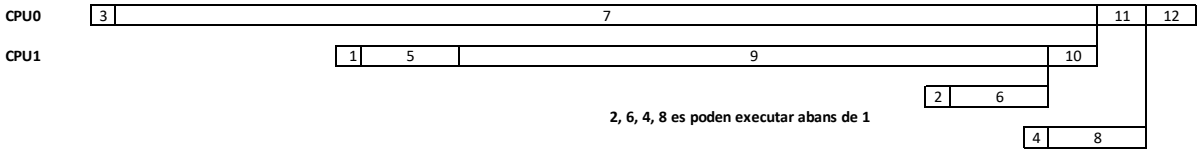
Conceptes i mètriques: TDG, T_1, critical path in TDG, T_∞, *Parallelism* and P_{min}, task granularity

Dubtes quizzes associat al vídeo lesson 2?



T1 =
Critical path =
T_∞ =
Parallelism =

Calcul de P_{min}?



En el diagrama temporal podeu observar que el camí crític ve determinat per 3-7-11-12, amb un cost (temps d'execució) superior al del camí mes llarg en quant a nombre de nodes (1-5-9-10-11-12). En el diagrama temporal cada cadena es col·loca el més tard que es pot executar, però es podria executar en qualsevol moment abans. Pel càlcul de P_{min} podeu veure que necessitem dos CPU, executant les cadenes 2-6/4-8/1-5-9-10 en una mateixa CPU i el camí crític en una altra.

Slide 9

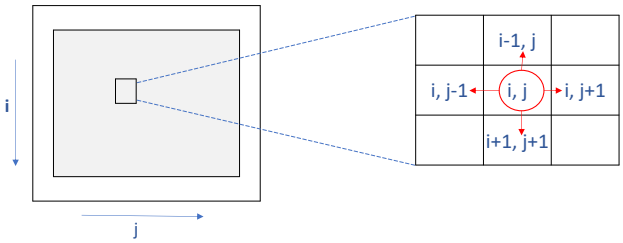
En la classe d'avui aprofundim en els conceptes i mètriques anteriors amb tres exemples concrets, que ens obriran les portes a descobrir els factors que poden determinar el paral·lelisme potencial que podem treure d'un algorisme. Començarem amb un de molt senzill, la suma dels elements d'un vector. Observem que la variable *sum* ens defineix una dependència de dades (precedència en l'execució de les tasques), de manera que T_∞=T_1 i per tant *Parallelism=1*.

Slides 10-11

Vol dir això que aquest problema no es pot paral·lelitzar? En aquesta slide mostrem que és possible reescriure el problema de forma diferent i obtenir més paral·lelisme. Per exemple, escrivint una versió recursiva del mateix problema, fent servir la tècnica de "divide and conquer". Un problema de mida *N* el dividim en *k* problemes de mida *N/k*; en el cas de la figura, *k=2*. En la slide 11 es mostra el TDG que s'obtindria i les mètriques corresponents, on ara veiem que T_∞ és proporcional al $log_2 n$ i per tant el paral·lelisme és proporcional a $n/log_2 n>1$ (*log* en base 2 per que hem fet un "divide and conquer" de 2).

Slide 17 – Jacobi relaxation

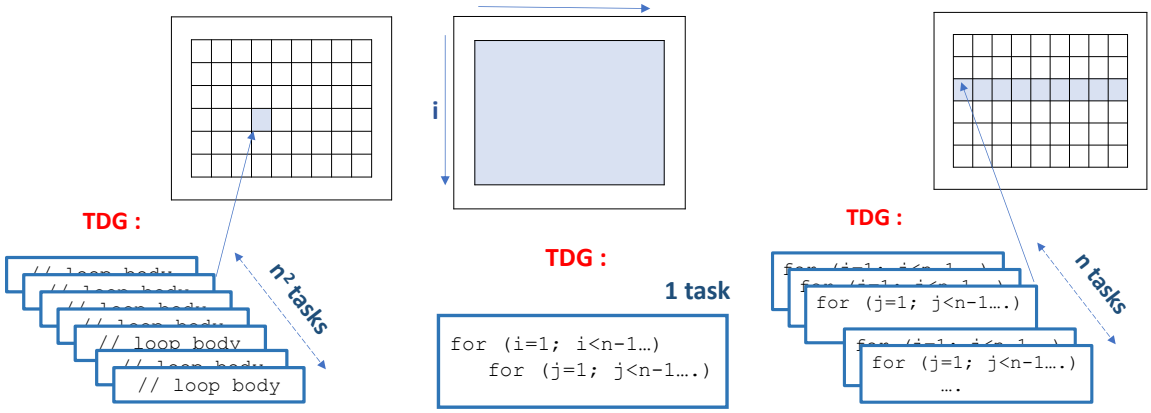
La slide 17 que ens donarà peu a treballar un segon aspecte també important a l'hora de decidir el particionat en tasques del nostre problema: el concepte de granularitat, que heu vist en la tercera part de la vídeo lesson 2. En la slide veiem una funció que calcula els elements d'una matriu *utmp* 2D de *nxn* elements a partir dels elements d'una altra matriu *u* també 2D de la mateixa mida, tal que cada element *utmp[i,j]* es calcula a partir dels 4 elements veïns de *u* i el mateix element.



Si volem paral·lelitzar aquest càlcul, què podríem definir com a tasca? Hi ha múltiples opcions ... en funció de la granularitat que volem donar a les tasques.

Slide 18

La slide mostra diferents granularitats de tasques, des de la descomposició més "fine grained" en la que es defineix una tasca per cada iteració del bucle més intern fins a la més "coarse grained" en la que es defineix una única tasca que engloba l'execució de totes les iteracions del bucle. Entremig diferents granularitats, cadascuna d'elles amb valors (expressions) diferents per les mètriques presentades, per exemple una tasca per calcular cada fila de la matriu (1 iteració del bucle i).

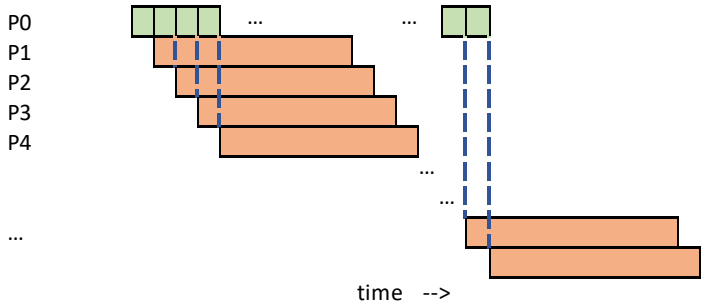


Observeu que la última fila de la taula en la slide representa el cas general (les altres files són casos particulars d'aquesta amb valors diferents de *r* i *c*). Important veure que per qualsevol definició de tasca, totes les tasques que s'obtindrien en el TFG són totalment independents.

Per tant, mirant a la darrera columna, una conclusió molt ingènua seria dir que com més fina sigui la descomposició (tasques més petites) més paral·lelisme estarem obtenint.

Slide 19

Aquest slide no té cap altre objectiu que tirar per terra la conclusió ingènua anterior ... basant-se en considerar una primera sobrecàrrega (overhead) en l'execució paral·lela: el overhead de crear les tasques, t_{create}. Ens anirà bé fer un diagrama temporal amb l'execució de les tasques per poder veure com influeix l'overhead i així obtenir l'expressió del T_∞



Observeu que el T_∞ ve determinat pel temps d'executar la última tasca creada, que ve precedida de la creació de totes les tasques prèvies. En concret, seria la suma de les dues últimes columnes de la taula que es mostra a la slide: *Task cost* més el *Task creation overhead*, que com veieu inclou el cost de crear totes les tasques.

$$T_\infty = r \cdot c \cdot t_{body} + (n^2 / (r \cdot c)) \cdot t_{create}$$

Observeu que a mida que *Task cost* disminueix el *Task creation overhead* augmenta. Per al cas simple de *r=c* (blocs quadrats)

$$T_\infty = c^2 \cdot t_{body} + (n/c)^2 \cdot t_{create}$$

podríem trobar el valor màxim de l'expressió anterior fent la seva derivada respecte a *c* i igualant a 0:

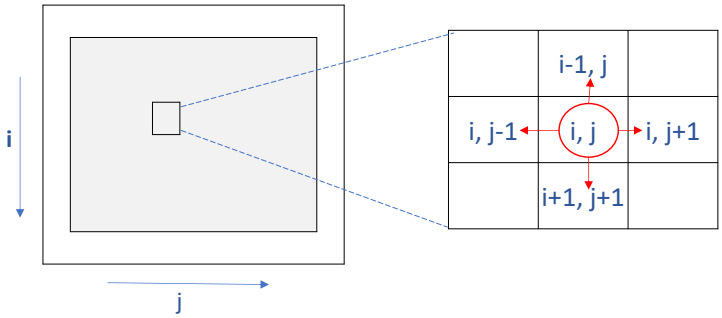
$$2 \cdot c \cdot t_{body} - 2 \cdot n^2/c^3 \cdot t_{create} = 0$$

D'on obtenim l'expressió per la mida del bloc c^2 (nombre total d'elements del bloc) que depèn de la *n* i la relació entre el cost d'executar el loop body i l'overhead de creació de cada tasca:

$$c^2 = n \cdot \sqrt{t_{create}/t_{body}}$$

Slide 20 – Gauss-Seidel relaxation

Quina diferència hi ha entre el codi d'aquest tercer exemple i el codi de l'anterior slide, el Jacobi? Ara hi ha dependències entre les iteracions del bucle, doncs s'està escrivint i llegint sobre la mateixa matriu *u*:



Per calcular l'element u[i][j] necessitem dos elements que s'han calculat en iteracions prèvies: l'element u[i-1][j] i l'element u[i][j-1]. De moment considerem un tipus de dependències, les que més endavant anomenarem "*flow dependence*" o "*Read-after-Write*": una tasca llegeix una variable que s'ha escrit en una tasca prèvia. Aqui veieu un exemple més senzill pel cas d'un vector:

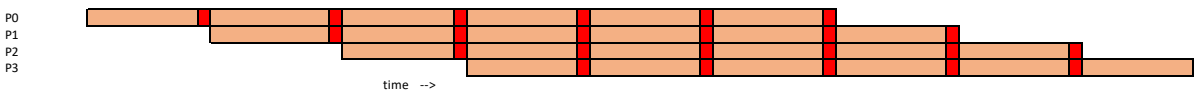
```
for (i = 1; i < n-1; i++) {
    ... = a[i-1] + ...;
    a[i] = ...;
}
```

A la iteració *i=23* escric *a[23]* que després es llegeix en la iteració *i=24*, al fer *a[24-1]*. En la propera classe veurem dos tipus més de dependències, per avui amb aquesta en tenim prou. I aquestes relacions d'ordre en els càlculs dels elements ens definiran les dependències entre les tasques que identifiquem. Suposant la definició de tasca com a bloc de *r·c* iteracions, una tasca qualsevol depèn de la tasca que hi just al damunt i la que te a la seva esquerra. El TDG que s'obtindria es mostra en la slide següent, pel cas particular de r i c que generen 4 tasques fila i 6 tasques columna, en total 24 tasques.

Slide 21

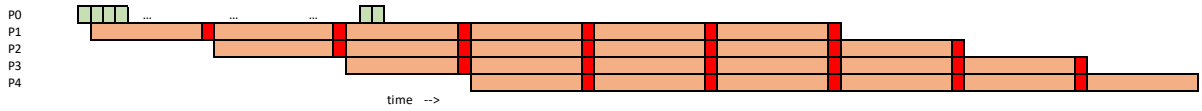
La slide mostra un TDG genèric pel cas de tasques definides com a blocs de *r* files consecutives x *c* columnes consecutives i les dependències que es creen entre les tasques. Per satisfer aquestes dependències, una tasca en acabar ha d'avisar a totes aquelles altres tasques que depenen d'ella. I això tindrà un overhead al que indiquem amb t_{synch} en la slide. Per calcular el T_∞ hem de considerar no només la suma dels pesos dels nodes que formen el camí crític, també hem d'afegir-hi les dependències en el camí crític, cadascuna amb el seu overhead t_{synch}.

Podem dibuixar el diagrama temporal per facilitar l'obtenció de l'expressió general del T_∞.



Cada ràfega de color beix es correspon amb l'execució d'una tasca. Podeu intuir quina es cadascuna si observeu les dependències entre tasques en el TDG. El diagrama també inclou els overheads de sincronització (ràfegues curtes de color vermell), algunes en el camí crític, altres que succeeixen en paral·lel amb les del camí critic. D'aquí podeu obtenir l'expressió que teniu a la slide.

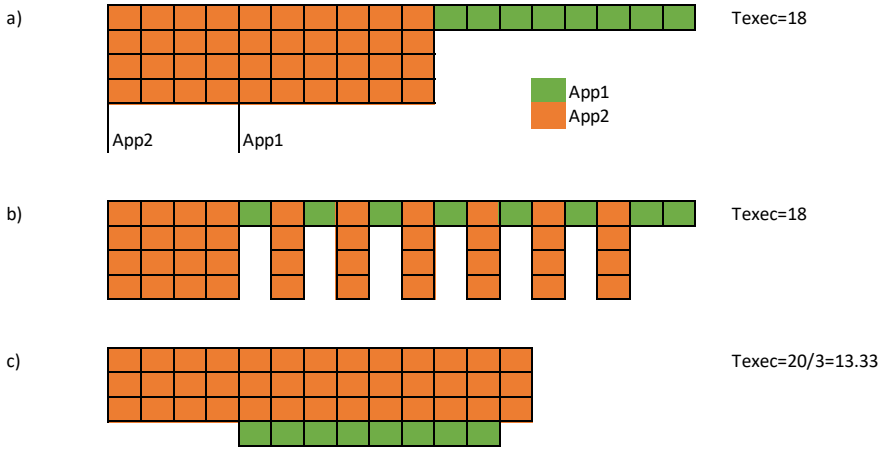
I com afectaria en aquest cas el overhead de creació de tasques? Forma part del camí crític com en el cas del Jacobi? Dibuixem el diagrama temporal corresponent per poder respondre a la pregunta:



Trobeu l'expressió per la mida òptima del bloc r·c (p.e. assumint r=c) tenint en compte els overheads t_{create} i t_{synch}.

Opcional: A Atenea trobareu un Excel on es representa el comportament de Gauss-Seidel, on podeu jugar amb els valors dels diferents paràmetres per veure on està el valor optim de **r** i *c*.

Problema 1 Tema 1



Problemes pendents de fer

Problema 2 del Tema 1.

Problemes a fer

Apartats a) i b) dels problemes 2 i 3. Es demana el TFG i les mètriques que hem treballat fins ara: T1, T_∞, parallelism i encara que no ho demanin, calculeu *el* P_{min} també. Important saber construir el TDG a partir del codi donat, en el que es fa servir la interfície de la eina Tareador amb la que treballareu en la segona sessió del Lab1. Però de cara a fer els problemes nomes cal identificar el que és una tasca, que queda delimitada per les crides a `tareador_start_task` i `tareador_end_task`.