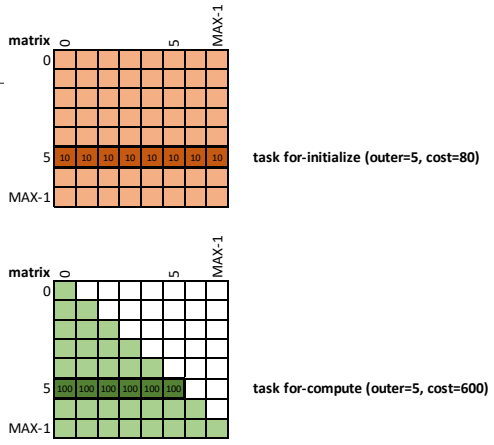
Problemes a fer

Problema 2

```
#define MAX 8
// initialization
for (outer = 0; outer < MAX; outer++) {
   tareador_start_task("for-initialize");
   for (inner = 0; inner < MAX; inner++)
      matrix[outer][inner] = inner;
   tareador_end_task("for-initialize");
   }

// computation
for (outer = 0; outer < MAX; outer++) {
   tareador_start_task("for-compute");
   for (inner = 0; inner <= outer; inner++)
      matrix[outer][inner] = matrix[outer][inner] + foo(outer,inner);
   tareador_end_task("for-compute");
   }
```

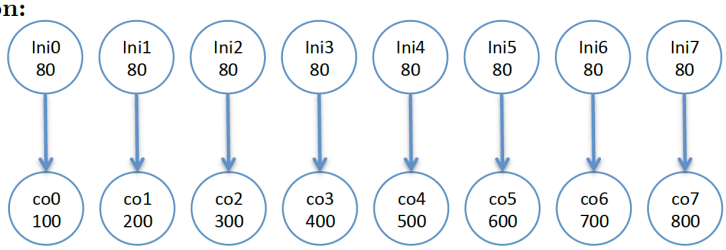


1) In the initialization loop the execution of each iteration of the internal loop lasts 10 cycles; 2) in the computation loop the execution of each iteration of the internal loop lasts 100 cycles; and 3) the execution of the foo function does not cause any kind of dependence.

Cada tasca en el primer bucle inicialitza una fila sencera de la matriu *matrix*; per exemple la tasca mostrada (outer=5) calcula tota la fila matrix[5][0..MAX-1], amb un cost de 80 time units. Totes les tasques del primer bucle són independents. D'altra banda, cada tasca en el segon bucle calcula (llegint i escrivint) una part d'una fila de la mateixa matriu; per exemple, la tasca mostrada (outer=5) llegeix i escriu matrix[5][0..5], amb un cost de (outer+1)*100=600. Igual que abans, totes les tasques del segon bucle son independents entre elles, però hi ha dependències entre les tasques dels dos bucles; en concret, les tasques amb el mateix valor d'outer són dependents, ja que en el primer bucle s'escriu la fila matrix[outer] i en el segon bucle es llegeix matrix[outer] (dependència Read-after-Write).

(a) Draw the task dependence graph (TDG), indicating for each node its cost in terms of execution time.

Solución:



(b) Calculate the values for T_1 and T_∞ as well as the potential parallelism.

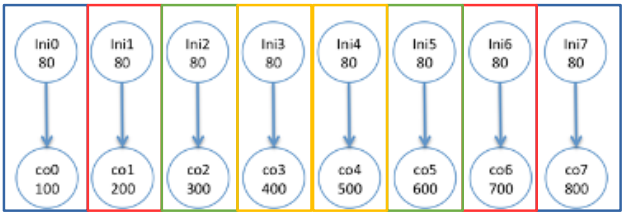
Solución: $T_1 = 4240$, $T_\infty = 880$, $parallelism = 4,81$

En aquest punt ens podem preguntar quin seria el valor de P_{min} per aquest TDG. Si arrodonim el valor de parallelism a l'enter superior ens donaria 5. Podeu comprovar que realment amb 5 processadors és possible d'aconseguir-ho (per exemple, P0: ini7-co7, el camí crític, P1: ini6-co6, P2: ini5-co5, ini0-co0, P3: ini4-co4, ini1-co1, P4: ini3-co3, ini2-co2).

(c) Calculate which is the best value for the "speed-up" on 4 processors (S_4), indicating which would be the proper task mapping (assignment) to processors to achieve it.

Solución: $S_4 = 4240/1060 = 4$, obtenido con la planificación siguiente:

Observeu que per tal de minimitzar el T_4 cal balancejar bé l'assignació de tasques a processadors, de manera que cada processador calculi:

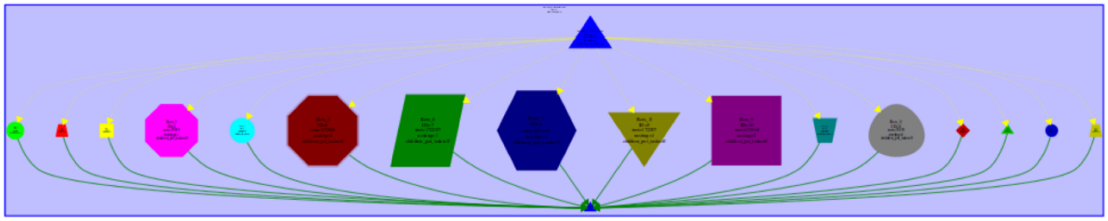


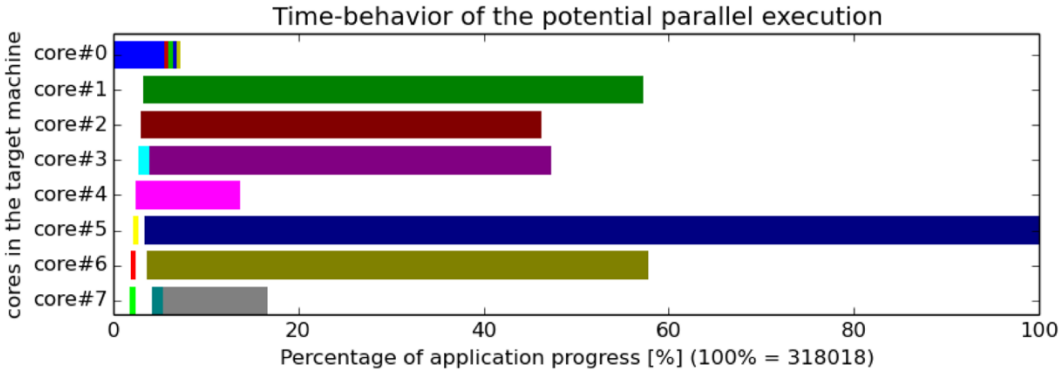
Resum vídeo lesson 3 y dubtes quizzes

Mètriques: scheduling (assignació de tasques) en P processadors → T_p, S_p i E_p

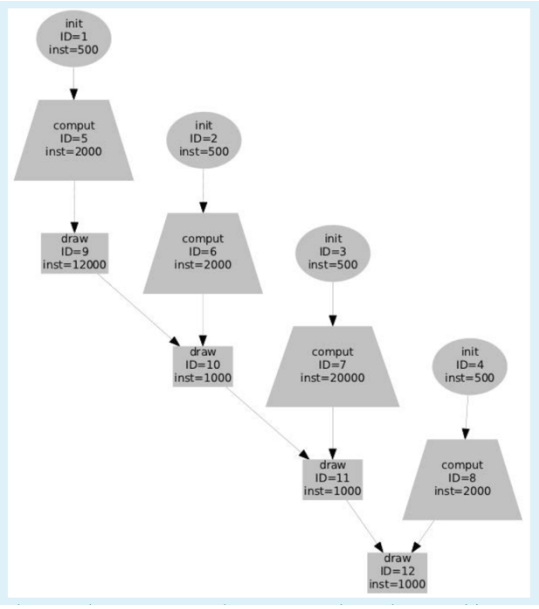
Per què no som capaços d'obtenir el speed-up ideal (S_p=P) amb P processadors?

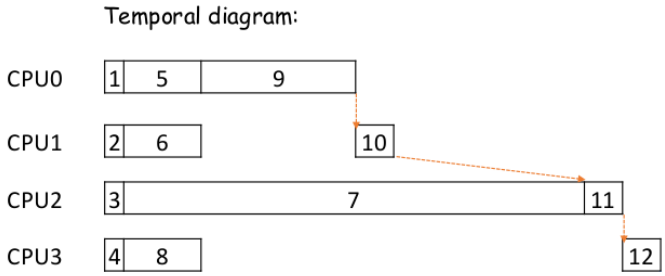
1. Desbalanceig de càrrega (exemple de Mandelbrot, primera part de la vídeo lesson 3)



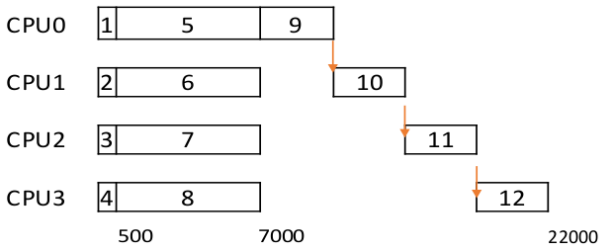


2. Dependències entre les tasques, camí crític, com vàrem veure en el cas de Gauss-Seidel vs. Jacobi en la darrera classe. També en l'exemple del quizz de la segona part de la vídeo lesson 3, abans de balancejar la càrrega teníem:



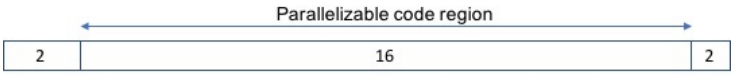


I després de balancejar (totes les tasques 5-8 amb pes 26000/4=6500) ens queda:

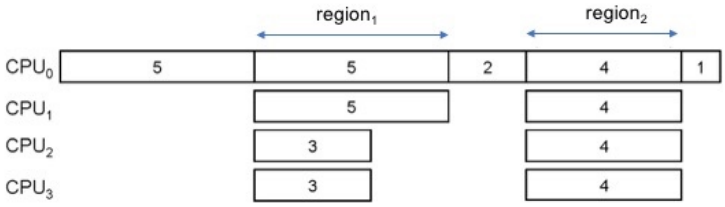


Com podeu veure, tot i estar ben balancejat el repartiment de feina, les dependències entre les últimes tasques no permeten aconseguir l'speed-up ideal.

3. Part seqüencial de l'aplicació que no es pot paral·lelitzar (Llei Amdahl, tercera part de la vídeo lesson 3): la fracció seqüencial d'un programa (1-ϕ) pot limitar l'escalabilitat? Amdahl law $S_\infty = 1/(1-\phi)$.



$\phi = 16/20=0.8 \rightarrow S_\infty = 1/(1-\phi) = 5.$

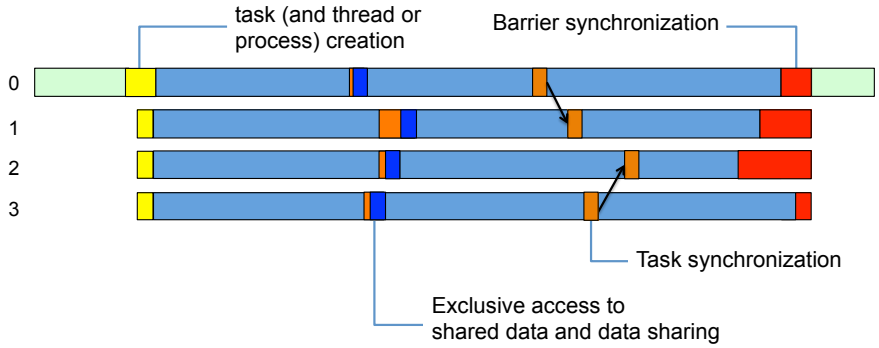


$\phi = (16+16)/(5+16+2+16+1)=32/40=0.8 \rightarrow S_\infty = 1/(1-\phi) = 5.$

Per calcular S_4 podem fer servir $S_p= 1/((1-\phi)+(\phi/p))$? No, doncs la primera regió NO escala de forma ideal. Però el podem calcular a partir del diagrama temporal:

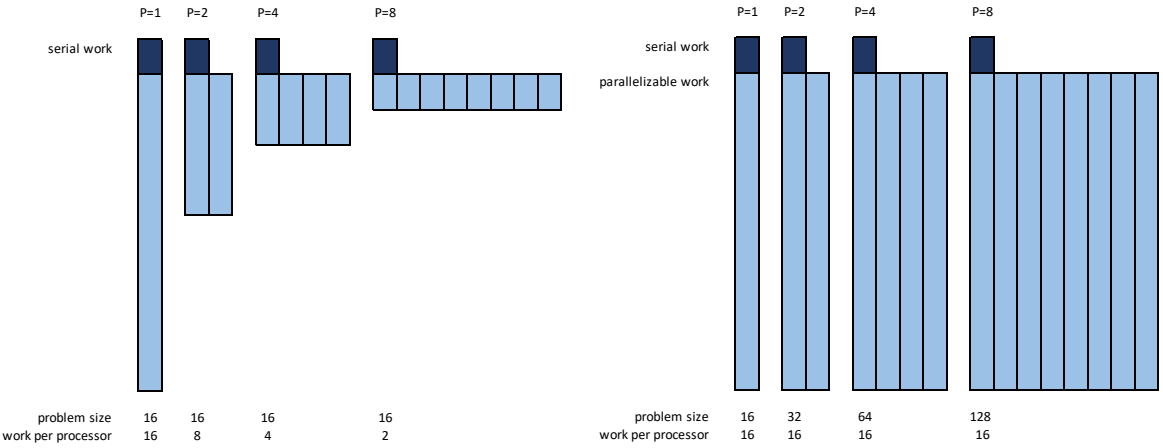
$S4 = T_1 / T_4 = 40 / (5+5+2+4+1) = 40 / 17 = 2.35.$

4. Overheads
    a. Task creation (t_{create})
    b. Task synchronization (t_{synch}) i protegir l'accés a variables compartides: slide 25



    c. Data sharing (**ho veurem avui**)

Strong vs. weak scaling (**who is who** in the two timelines below? time goes in vertical, up down ...)

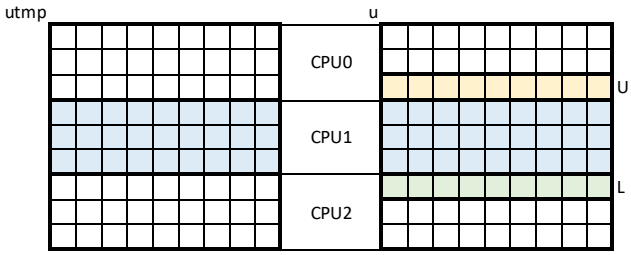


Slides 28-30

Model de cost per accessos locals (overhead 0) i accessos remots (overhead $T_{access}= t_s+m \cdot t_w$), on t_s és el temps d'iniciar l'accés remot (start-up time) i t_w és el temps de accedir a cada paraula (transfer time). Usualment $t_s >> t_w$, per tant: 1) és interessant minimitzar el nombre d'accessos remots i 2), si es possible, que en cada accés remot s'accedeixin al màxim numero possible d'elements. Restricció addicional: en cada instant de temps cada processador pot fer un accés remot i servir un altre accés remot.

Slides 31-34 (Jacobi solver)

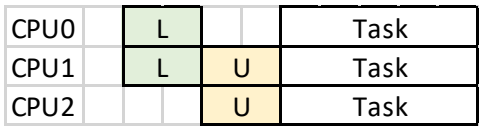
Task definition: n / P consecutive iterations of i loop, being P the number of processors. L'altre dia vàrem veure que no hi ha dependències entre les tasques. En altres paraules, una tasca no necessita accedir al resultat d'una tasca prèvia. Però observeu que una tasca per calcular les seves files d'utmp (blau a l'esquerra, figura de baix) necessita accedir a les seves files de la u (blau a la dreta), a la última fila del processador de dalt (fila en groc, degut al accés u[i-1][j] en calcular la seva primera fila, etiqueta U en el diagrama temporal posterior) i a la primera fila del processador de baix (fila en verd, degut al accés u[i+1][j] en calcular la seva ultima fila, etiqueta L en el diagrama temporal posterior).



Observeu que els accessos u[i][j-1] i u[i][j+1] són accessos locals. Per tant, a l'expressió de T_p li hem d'afegir el overhead d'accedir a aquestes files.

1. Quan accedir a aquestes dades? Abans de començar el càlcul, així eliminem la "anti-dependence" i les tasques poden anar en paral·lel.
2. Com accedir a aquestes dades? Un únic accés per accedir a les dades de cada processador veí, així només paguem una vegada el t_s.

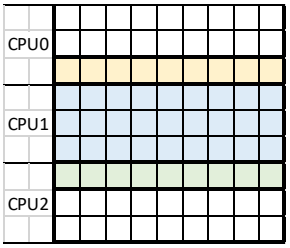
Per tant, el diagrama temporal seria:



A les slides trobareu el desenvolupament de l'expressió per T_p.

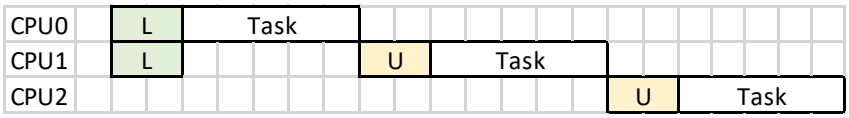
Slides 35-37 (Gauss-Seidel solver, primera definició de tasques)

Recordar de la classe de l'altre dia que la mateixa definició de tasca en el Gauss-Seidel no dona cap paral·lelisme donades les dependències que tenim entre tasques.



La CPU1 no pot començar a executar el seu bloc de files fins que la CPU0 acabi de calcular la seva última fila. I la CPU2 no pot començar a executar el seu bloc de files fins que la CPU1 acabi de calcular la seva última fila. Per tant, l'execució queda seqüencials més el overhead

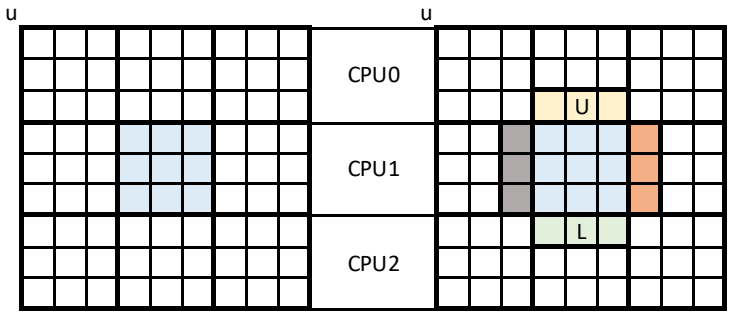
addicional d'accedir a aquestes files abans de començar amb l'execució de la tasca, tal com mostra el diagrama temporal:



Els accessos L segueixen existint per tal d'accedir a la primera fila del processador seguent. Aquest cas el teniu desenvolupat a les slides.

Slides 38-41 (Gauss-Seidel solver, segona definició de tasques)

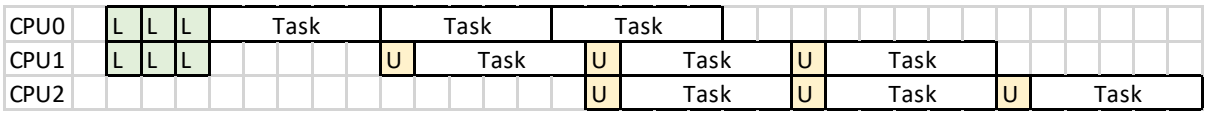
Task definition: block of n / P by c consecutive iterations of i and j loops, respectively; P is the number of processors. De la classe de l'altre dia ja vàrem veure que amb aquesta definició de tasca, tot i que tenim dependències amb la tasca de dalt i la de l'esquerra, sí que tenim paral·lelisme a explotar. Per tant ara ens trobem que una tasca SI necessita accedir al resultat de tasques prèvies.



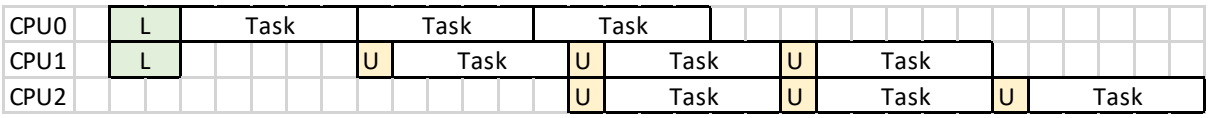
Anem a veure amb detall. Una tasca per calcular el seu bloc de files/columnes d'u (blau a l'esquerra, figura de dalt) necessita accedir:

- Al seu bloc de files/columnes de la mateixa u (blau a la dreta). Tots aquests accessos són locals.
- Al tros de fila del processador de dalt (tros de fila en groc, degut a l'accés u[i-1][j] en calcular el tros de la seva primera fila, etiqueta U en el diagrama temporal posterior). Observeu que és un accés a dades calculades per una tasca, té overhead. L'accés remot a U s'ha de fer en acabar l'execució de la tasca prèvia, accés amb mida c.
- Al tros de fila del processador de baix (tros de fila en verd, degut a l'accés u[i+1][j] en calcular la seva última fila, etiqueta L en el diagrama temporal posterior). Observeu que és un accés a dades remotes que no les ha calculat una tasca prèvia, si té overhead. L'accés remot L s'ha de realitzar abans de començar l'execució de les tasques assignades al processador.
- Al tros de columna del processador de l'esquerra (tros de columna en gris, degut a l'accés u[i][j-1] en calcular el tros de la seva primera columna). Observeu que és un accés local, amb dependència amb una tasca prèvia que ha calculat el mateix processador; no té overhead doncs és un accés local.
- I al tros de columna del processador de la dreta (tros de columna en taronja, degut a l'accés u[i][j+1] en calcular el tros de la seva ultima columna). Observeu que també és un accés local, en aquest cas sense dependència amb una tasca prèvia; així tampoc té overhead doncs és un accés local.

El diagrama temporal seria:



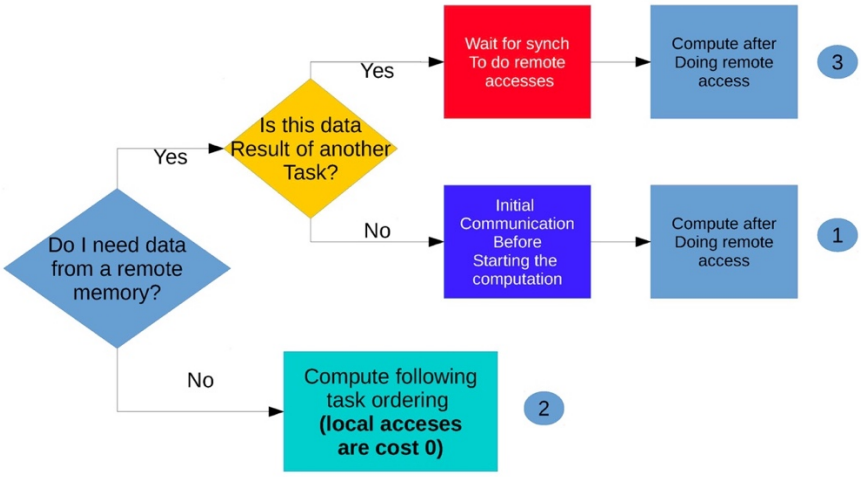
on veiem que cada processador fa múltiples accessos L al següent processador (excepte l'últim); observeu per tant que aquí tenim l'oportunitat de que tots els accessos L que ha de fer un mateix processador s'agrupin en un únic accés remot, minimitzant així l'overhead (queda un únic accés remot de mida n enlloc de n/c accessos remots de mida c, es a dir $(t_s+n\cdot t_w)$ enlloc de $(n/c)\cdot(t_s+c\cdot t_w)$). Així el diagrama temporal seria:



A les slides trobareu el desenvolupament de l'expressió per T_p.

De fet els accessos u[i+1][j] i u[i][j+1] provoquen dependències? Clarament no són una *"Read-after-Write dependence"* ... són una *"Write-after-read dependence"* que en la literatura anomenen també *"false dependences"* o també *"anti-dependences"*, com a contrapartida a les *"true dependences"*. En el cas de u[i+1][j] la anti-dependència desapareix quan fem la còpia inicial de la fila L. En el cas de u[i][j+1] no necessita accés remot doncs són dades locals.

Resum: "Quan cal fer accessos remots?"



Slide 42

És possible trobar un valor òptim per la mida del bloc (valor de c) de manera que es minimitzi el T_p. Similar al que vàrem fer l'altre dia quan vàrem considerar el t_{create}. Derivar l'expressió de T_p i igualar-la a zero per trobar el seu mínim.

Problemes pel proper dia

Problema 5 sobre Llei d'Amdahl
Problema 9 (apartats b, d i e) sobre overheads de sincronització
Problema 14 sobre overheads de data sharing