Guió i material per classe no presencial PAR grup 20
Dilluns 15/2/2021, 10:00-12:00

Repàs vídeo lesson 1

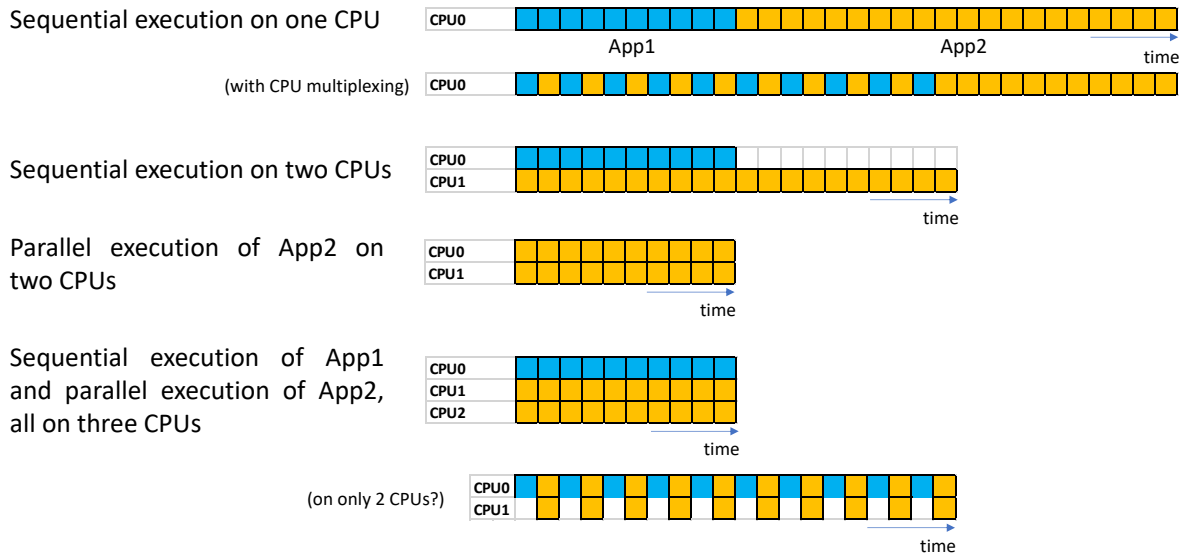
Què és paral·lelisme i perquè és necessari conèixer-lo per tal de poder explotar eficientment les arquitectures que formen els computadors d'avui en dia?

Objectiu: reduir el temps d'execució d'un programa basant-se en repartir les seves instruccions entre múltiples processadors. Més endavant veurem que també aplica quan volem augmentar la mida del problema a resoldre mantenint el temps d'execució.

Exemples de màquines (slide 8) i efecte en un programa concret (slide 9).

Dubtes quizz associat a la vídeo lesson 1?

> Let's consider two applications: App1 and App2. Assume that App1 and App2 execute N1=10 and N2=20 instructions, respectively, and that processors are able to execute 1 instruction per time unit (F=1).



Slide 10

Què és concurrència? Dividir l'execució d'un programa seqüencial en tasques, que seran executades de forma multiplexada en un sòl processador. Perquè ho hem de fer? Doncs per què la funcionalitat de l'aplicació ho requereix. Per exemple, una aplicació tipus client/servidor.

Slides 11-13

En concret, imagineu-vos l'aplicació d'un banc que rep peticions de clients (fer pagament de rebuts, transferències entre comptes bancaris, ...). Dos tipus de tasques en general: tasca client (C) que s'encarrega de rebre la petició que un client vol fer (i poden haver-hi múltiples clients simultàniament); tasca servidor (T) que s'encarrega de processar la petició del client (una instància per cada petició de client C).

En la slide 11 totes les tasques es fan una darrera l'altre, un cop acabada una tasca completa en comença un altre → temps d'espera a ser ates i temps de finalització pot ser molt alt. En

la slide 12, quan un client es connecta s'inicia la seva tasca C, multiplexant l'execució de la tasca T que s'estava executant-se en aquell moment → temps de finalització pot ser molt elevat, sense resposta fins que no acabin totes les peticions de clients anteriors. Finalment, en la slide 13 l'execució de les tasques C i T es van multiplexant, donant la sensació que tots els clients estan sent atesos. **En qualsevol cas, el temps d'execució total és la suma dels temps d'execució de totes les tasques, s'està processant 1 petició per unitat de temps, encara que hi ha múltiples que estan simultàniament actives.**

Dubte: Així, el temps d'execució concurrent multiplexat serà el mateix que el temps d'execució seqüencial? Si no considerem cap mena d'overhead en la multiplexació de tasques, el temps de tot el programa sí que serà el mateix. Però ja hem vist que amb la multiplexació els clients es veuen atesos i les seves transaccions comencen a executar-se tan aviat com és es possible; això sí, compartint la CPU entre totes elles (i per tant executant-se a una velocitat més lenta). Quins overheads a part del propi overhead del SO en fer la multiplexació? Penseu per exemple que les dades que una tasca té a la jerarquia de memòria es perdran (en part o totalment) pel fet que la nova tasca que ocupa la CPU utilitza la mateixa jerarquia de memòria; aquesta pèrdua de localitat pot provocar un cert augment en el temps d'execució.

Slides 14-15

Amb l'execució paral·lela dotem a l'execució concurrent amb múltiples processadors (P), que poden executar en el mateix instant de temps tantes tasques com processadors. Amb el paral·lelisme estem reduint el temps d'execució de l'aplicació, en el cas de l'aplicació client/servidor, estem donant resposta i finalitzant les peticions en un temps menor. Ara el temps d'execució total es redueix "potencialment" per P, o en altres paraules, estem processant més peticions per unitat de temps.

Slide 16

Aquesta idea de multiplexació de tasques aplicada a multiplexació de múltiples aplicacions és el que es coneix com a "throughput computing", que també us hem comentat en la vídeo lesson 1. Cadascun dels programes pot ser seqüencial o paral·lel, requerint per tant 1 o més d'un processador per ser executats. El sistema operatiu (SO) intentarà repartir els processadors de la forma més equitativa possible entre aplicacions. En el cas d'aplicacions paral·leles, suposarem que el SO els hi dóna el nombre de processadors que necessiten per ser executades. Observeu que throughput "per-se" no requereix canvis en l'aplicació; concurrència/paral·lelisme impliquen canvis en les aplicacions.

Slide 18

Anem a veure tres exemples diferents de tasques T (transaccions), interactuant amb l'estructura de dades del banc que guarda la informació dels comptes corrents. Per simplicitat, per cada account tindrem el seu *id* i el *balance* (o saldo disponible).

Slide 19

El primer exemple és pel cas que la tasca T sigui "treure/dipositar diners en el compte corrent". Aquí teniu un esbós del codi de la tasca: en primer lloc es valida si hi ha prou diners a l'*account* per tal de treure un missatge d'error o per procedir a fer la transacció

corresponent (sumar o restar val a balance). Observeu el `#pragma omp task` que especifica que és una tasca, model de programació OpenMP que farem servir a PAR.

Slide 20

Què pot passar si simultàniament s'executen 2 tasques T que volen fer aquesta mateixa transacció?. Si volen treballar amb dos *accounts* diferents ($id_1 \neq id_2$) no hi haurà cap problema. El problema el tenim si volen operar amb el mateix compte corrent ($id_1 = id_2$).

Slide 21

Apareix el que s'anomena una **data race** (condició de carrera). Situació mostrada en la slide, resultat incorrecte doncs les dues tasques modifiquen la mateixa variable *acc.balance*.

Slide 22

Hem de protegir l'actualització del *balance* per tal d'assegurar que només una tasca ho fa en cada instant de temps. Ja veurem que els `locks` són un mecanisme que ens permetran assegurar això. Només una tasca executa l'actualització en cada instant de temps; la resta de tasques que ho volguin fer, s'esperaran.

Slide 23

El problema persisteix, ja que la lectura del *acc.balance* en el moment de fer la comprovació no està protegida. Fàcil solució ...

Slides 24-25

... simplement protegint tota la transacció amb el `lock`. Fixeu-vos que el `lock` està associat al account en concret (*acc.lock*). Així, el `lock` serialitzarà les transaccions només si volen treballar sobre el mateix *account*; si no s'executaran en paral·lel.

Slide 26

Segon exemple: la transacció ara consisteix a fer una transferència bancària entre dos *accounts* diferents: de l'*account* `from` hem de treure diners que ficarem a l'*account* `to`. Per evitar les possibles data races en els accessos farem servir dos `locks`, un per protegir cadascun dels *accounts*.

Slides 27-28

Però que passaria si simultàniament es volen fer dues transferències creuades, tal com mostra la slide? Aquí ens apareix un problema de **deadlock**, ho podeu veure en la slide 28, en la que cadascuna de les tasques queda bloquejada esperant que l'altre alliberi el `lock` que necessita.

Slide 29

Podem arreglar el problema si sempre agafem els `locks` en el mateix ordre, per exemple primer el `lock` de l'*account* amb identificador major.

Slide 30

Altres problemes relacionats amb la concurrència que podreu veure el vídeo opcional que trobareu a Atenea.

Slide 31

Finalment, el darrer exemple per avui és el mostrat a la slide. Imagineu-vos que cada dia el banc té de treure unes estadístiques que ha d'enviar a hisenda per demostrar que està "sà". Per fer un exemple senzill, imagineu que ha de calcular l'interès que hauria de pagar a tots els seus clients si l'endemà decidissin treure els seus diners del banc.

Slide 32

Si resulta que això triga molt de temps perquè el banc té centenars de milers d'*accounts*, podem fer el càlcul en paral·lel per tal de reduir el temps d'execució. Cada processador calcularà la suma parcial per una part dels clients, que al final s'hauran de sumar.

Slide 33

De nou aquí us mostrem el codi en OpenMP, simplement per veure el fàcil que pot arribar a ser expressar el paral·lelisme en el nostre programa. Un simple `#pragma omp parallel for` per dir que volem anar en paral·lel repartint les iteracions del bucle entre tots els processadors disponibles.

Slide 34

L'execució paral·lela en aquest cas fent servir el servidor boada del departament d'AC, que fareu servir per fer les pràctiques de PAR, ja que ens mostra un comportament quasi ideal. Quants més processadors més ràpid anem (quocient entre el temps seqüencial i el paral·lel).

Slide 35

Però vaja, tenim tot el quadrimestre per complicar-nos la vida programant en paral·lel i entenent els factors que faran que el comportament de les aplicacions paral·leles no sempre sigui tant ideal com el mostrat en la slide anterior. Aquí simplement enumerem alguns dels problemes que ja anirem veient al llarg del quadrimestre.

Finalment comentem les slides del Tema 0 de presentació de l'assignatura, simplement per veure els aspectes rellevants de com anirà el curs:

- Objectius de l'assignatura: fer models simples que ens permetin entendre el rendiment esperat de les aplicacions paral·leles, les arquitectures de memòria compartida que considerarem durant tot el curs i la programació de programes paral·lels, la seva execució i anàlisi fent ús del model de programació OpenMP (amb dues estratègies: task vs. data decomposition)
- Temari: 5 temes, cadascun amb una durada de 3 sessions, excepte el tema 1 que el fem tot avui.
- 1 classe de teoria (online) i 1 de laboratori (presencial) setmanals.
- Metodologia flipped-classroom, no és un invent del covid, ja ho estàvem fent a PAR des de fa temps. Vídeos i quizzes online, com els que heu fet per la Unit 1 durant la setmana anterior. Molt bona participació. Us animem a continuar així.

- Control durant la setmana que la FIB dedica a la realització de controls. En concret per PAR el dia 21 d'abril, de 8:00 a 9:30. Control presencial.
- Examen final el 16 de juny, de 8:00 a 11:00. Examen presencial.
- Laboratoris ja us explicaran els professors de laboratori, però bàsicament tindrem 5 pràctiques de laboratori, a fer en grups de 2 estudiants. Totes les practiques ens ocuparan 3 setmanes, excepte la 2 i la 3 que les farem en 2 setmanes. Estudiants repetidors que van tenir una nota igual o superior a 7 podran fer un itinerari alternatiu que els ocuparà les 5 primeres setmanes de curs; si superen la practica assignada, ja esta; si no, continuaran amb les practiques regulars.
- Tot el material de l'assignatura en anglès, incloent-hi enunciats d'exàmens. Resposta en català/castellà o anglès.
- Avaluació:
    - Control P (presencial), temes 1, 2 i 3.
    - Final F, tots els temes.
    - Control (35%) i final (65%).
    - 75% control/final (teoria), 25% laboratori.
    - Si s'aprova l'assignatura, fins a un 10% de la nota obtinguda addicional d'acord amb la participació a Atenea: resolució de quizzes abans del deadline previst per cada unitat (és a dir, abans de la classe de teoria corresponent)
- Detalls de part de pràctiques ja ho farà el vostre professor de laboratori