Material complementario 24/03/2020

Arno Formella

Este documento recoge más o menos lo que hubiese contado en clases presenciales a lo largo de la exposición paulatina de las transparencias. En clase normalmente descubro diferentes partes del contenido de cada transparencia poco a poco, para no sobrecargar con información de golpe, y para mantener un hilo de pensamiento. En las transparencias distribuidas todo viene de golpe, por eso recomiendo trabajar con ellas lentamente y ver este material complementario a la par.

Estarán incluidas preguntas (a veces sin respuestas) para animar a la reflexión sobre el tema y a la búsqueda de información adicional. Admás proporciono más referencias a la red y la bibliografía.

Ojo, las páginas mencionadas aquí siempre se refieren a los tochos que publico para cada semana. Puede ser (y es casi seguro) que en el documento global (que desarrollo en paralelo) la enumeración de las página va a variar, ya que se añaden transparencias en otros lugares.

P144

Es una transparencia para pensar.

Fijaros que queremos multiplicar dos números con n dígitos (aquí hay dos con 12 dígitos, pero podrían ser millones).

Todos hemos aprendido como multiplicar estos números a mano, que resulta en una algoritmo del orden $O(n^2)$ ya que primero se multiplica cada dígito de un factor con el otro factor, y todos los productos parciales se suma.

Y si queremos hacerlo en paralelo... ¿Cómo realizamos los desafios mencionados antes (algoritmo, divisón trabajo, distribución datos, sincronización, comunicación...)?

En el algoritmo de ordenación por contado hemos logrado, que los procesos (menos en los casos de reducción) pueden trabajar independientemente cada uno con sus propios datos. Pero aquí, parece más complejo...

Imaginaros que cada dígito de los dos factores está en un proceso, y queremos hallar el producto con mensajes *whatsapp* de forma correcta y eficiente...

P145

El resultado presentado fue calculado con *python* que internamente contiene un algoritmo de multiplicación de enteros con precisión muy alta.

No entramos en detalles de algoritmos de multiplicación, solamente quiero mencionar algunos aspectos interesantes:

- El algoritmo que todos aprendemos en la escuela no es un algoritmo eficiente (ya que necesita $O(n^2)$ operaciones.
- Desde hace unos 50 años se conoce un algoritmo (Schönhage-Strassen) que multiplica en $O(n \log n \log\log n)$ y los autores han dicho que debe ser posible hacerlo en $O(n \log n)$.
- Hace ahora más o menos un año que Harvey y van der Hoeven por fin han comprobado que tenían razón: han publicado un algoritmo que multiplica números de n digitos en tiempo $O(n \log n)$.
- Si echamos suficientes procesos a trabajar podemos convertir el algoritmo clásico en un algoritmo que multiplica (en paralelo) en tiempo $O(\log n)$ (pero no será eficiente si lo simulamos con un solo proceso, ya que necesitaría tiempo $O(n^2)$). Con estos algoritmos funcionan las CPUs (que en principio echan transistores a trabajar para lograr una multiplicación rápida).
- Según yo sepa todavía nadie a logrado un algoritmo de multiplicación que cumple con un tiempo de calculo de $O(n/p \log n)$ para cualquier número p de procesos.
- Tal comportamiento de algoritmos se llama "eficiente respecto al trabajo en paralelo" (*work-efficient parallel algorithms*), es decir, p procesos son de verdad p-veces más rápido que un solo proceso en el algoritmo más eficiente conocido.

Bueno, dejamos este excurso a la teoría de los algoritmos y hacemos un ejercicio de multiplicar dos números con una CPU super simple para aprender ciertos aspectos de la programación concurrente.

P146

Es decir, dado que solamente podemos sumar (y con eso subtraer) necesitamos un algoritmo basado en estas dos operaciones.

Observa que por el momento solamente estamos interesados en multiplicar números positivos.

Analiza bien el algoritmo propuesto:

- Accumula en la variable `r` tantas veces la variable p como indica la variable `q`.
- Pero ¿Cómo comprobamos que de verdad funciona correctamente?

P147

Convéncete que es muy bueno saber que tanto el algoritmo como el programa están correctos.

Hablando del programa tal como presentado antes, asumiendo que las variables `p` y `q` estén correctamente inicializados, deberíamos llegar a la instrucción `g:` con el resultado en `r`.

Parece obvio, pero en el fondo no lo es.

¿Qué método matemático usamos para la comprobación?

P148

Primero: cada paso debe funcionar correctamente.

Normalmente trabajamos con ordenadores que funcionan, y ejecutan las instrucciones de forma esperada y correcta.

(Pero no es siempre asi: fíjate en el fallo de división de un procesador de Intel en los años 90 `https://en.wikipedia.org/wiki/Pentium_FDIV_bug`)

O piensa que estás actuado con todo una cadena de hardware y software (CPU, memoria, compilador, ensemblador, sistema operativo...), en principio, en cualquier eslabón puede estar escondido un fallo.

Bueno, asumimos que los pasos individuales (en nuestro pseudo-código cada línea) están funcionando correctamente.

Nos armamos con la herramienta de *inducción* para realizar la comprobación formal.

P149

Encontrar y formular una *invariante* es un paso crucial para este tipo de comprobación.

La idea es encontrar justamente una propiedad del estado de la máquina (en nuestro caso los posibles valores que pueden tener las variables usados en el algoritmo) que **no** cambia mientras estamos ejecutando el programa.

La inducción ahora es fácil:

$i = 0$ (es decir, estamos la primera vez en la línea `c:`)

tenemos para las tres variables (en ronda 0):
$p_0 = p$, $q_0 = q$, $r_0 = 0$
entonces para la invariante:

$$r_0 + p_0 \cdot q_0 = 0 + p \cdot q = p \cdot q$$

entonces el caso base es correcto.

Asumimos entonces que la hipótesis es correcta (es decir, hemos pasado i-veces por el bucle):

$$r_i + p_i \cdot q_i = p \cdot q$$

ejecutamos un paso del bucle (es decir, las instrucciones `d:`, `e:`, `f:` y `b:`), llegamos de nuevo a `c:`

y tenemos:

$$\begin{aligned} r_{i+1} + p_{i+1} \cdot q_{i+1} &= r_i + p_i + p_{i+1} \cdot q_{i+1} && (1)\\ &= r_i + p_i + p_i \cdot q_{i+1} && (2)\\ &= r_i + p_i + p_i \cdot (q_i - 1) && (3)\\ &= r_i + p_i + p_i \cdot q_i - p_i && (4)\\ &= r_i + p_i \cdot q_i && (5)\\ &= p \cdot q && (6) \end{aligned}$$

la secuencia de estas ecuaciones se da por: (1) hemos ejecutado línea `d:`, (2) no se modifica la variable `p` en el bucle, (3) hemos ejecutado línea `e:`, (4) y (5) operaciones de equivalencia en la ecuación, (6) uso de la hipótesis.

Entonces hemos comprobado la invariante.

Falta por comprobar que lleguemos a la línea `g:`.

Pues la variable `q` contiene un valor positivo entero al principio, solamente se esta decrementando por uno en cada paso por el bucle (línea `e:`), entonces en algún momento se tiene que llegar a cero, entonces el programa termina el búcle en línea `c:` y llega a línea `g:` con el resultado en r ya que si $q_{i+1} = 0$ entonces según invariante comprobada $r_{i+1} = p \cdot q$ (obviamente también en el caso base si $q = 0$ desde el principio).

P150

Pues, tenemos un algoritmo de multiplicación secuencial correcto.

Producimos un algoritmo paralelo con dos procesos P0 y P1 simplemente duplicando el código en los dos lados y actuando sobre variables compartidas.

La pregunta escencial es: ¿es correcto, funciona?

Lo primero que notamos es: pues quizá no, pero quizá sí, y como comprobamos el úno o el otro...

P151

Antes de entrar en más detalles, implementamos el pseudo-código en Java con clases propias para los números enteros (basicamente para "enchufar" diferentes implementaciones para tales enteros de forma simple, aunque sin el concepto de herencia, si alguien quiere hacerlo: pues adelante...)

P152

Una clase para enteros que realiza justamente lo que necesitamos. (libremente según el dicho: líneas de código que no existen tampoco están mal :-)

P153

Las variables compartidas (nota que Java trabaja con referencias!) en una `Runnable` para implementar para cada hilo su programa (y todos serán iguales como era la idea).

El constructor inicializa las variables y un identificador para cada hilo.

P154

Observa que hemos implementado exactamente el bucle de nuestra multiplicación (identifica las líneas entre Java y pseudo-código!).

Hemos usado el mecanismo de excepciones (no esperamos en principio ninguna), y usamos el bloque final para que el hilo diga que haya terminado.

P155

Tratamos los parámetros por línea de comando que serán los factores p y q más el número de procesos participantes.

Inicializamos las variables (como exigido en el algoritmo).

P156

Creamos los n hilos, los lanzamos, y esperamos hasta todos hayan terminado (eso ya vistéis en prácticas).

P157

Última parte del programa principal visualiza el resultado r, y contiene los bloques necesario para la excepción global.

P158

Importante: Implementa el programa de Java! (creo que basta que hagas los copy&paste de las tranparencias :-)

Ejecuta el programa con un hilo (n=1)!

¿funciona?

En caso que sí; ¿por qué funciona?

Ejecuta el programa con dos hilos (n=2)!

¿funciona? ¿funciona siempre igual? o lo hace a veces, o muchas veces, o pocas veces, algo raro?

En caso que sí; ¿por qué funciona? En caso que no; ¿por qué no funciona?

En mi opinión es importante que realices las tareas para desarrollar una posición crítica al asunto.

En adelante vamos a discutir y remediar los problemas que tiene!