Parcial teórico 1

jueves, 11 de marzo de 2021 11:59

Parte A

1- Defina programa distribuido, programa paralelo y programa concurrente.
Un programa paralelo es la ejecución concurrente en múltiple procesadores, con el objetivo de incrementar la performance.
Un programa distribuido es un programa concurrente que se comunica mediante el pasaje de mensajes. Suele ejecutarse en arquitecturas de memoria distribuída, pero también lo puede hacer en memoria compartida.
Programa concurrente: un programa concurrente consiste en un conjunto de "programas secuenciales" que pueden ejecutarse concurrentemente en el tiempo, especificando además como se van a comunicar y sincronizar estos procesos.

2- Defina sincronización entre procesos. Describa los mecanismos de sincronización que conozca.
La sincronización entre procesos consiste en la posesión de información de un proceso para coordinarse con los demás.
Mecanismos de sincronización:

- Exclusión mútua: asegurar que sólo un proceso tenga acceso a un recurso compartido en un instante de tiempo.
- Sincronización por condición: bloquea la ejecución de un proceso hasta que se cumpla una condición dada para asegurar el orden temporal necesario.

3- Defina la instrucción Test & Set. ¿Cuál es su utilidad para la sincronización?
La intrucción TS pone el boolean (que generalmente es el lock) recibido en true y retorna el valor original del mismo. Sirve para hacer una implementación atómica del await de grano grueso, se utiliza en el algoritmo de spin lock, aunque no es la mejor opción. Es una intrucción que viene en la mayoría de los procesadores.
bool TS (bool ok);
{ <bool inicial = ok; ok = true; return inicial;> }

4- ¿En qué consiste la propiedad de “A lo sumo una vez” y qué efecto tiene sobre las sentencias de un programa
concurrente?
La propiedad a lo sumo una vez dice que, para que la ejecución de una asignación u expresión sea atómica, debe cumplir con esta propiedad. Plantea que:
Siendo una sentencia x = e, satisface esta propiedad si:

- e contiene a lo sumo una referencia crítica y x no es referencia por otro proceso
- o
- e no contiene referencias críticas, entonces x puede ser leída por otro proceso

Una expresión e cumple con la propiedad si no contiene más de una referencia crítica.
Una referencia crítica es una referencia a una variable modificada por otro proceso.

5- ¿Qué significa que un problema sea de “exclusión mutua selectiva” (EMS)? Describa el problema de los 5 filósofos.
¿Este problema es de EMS? ¿Por qué? Si sólo hubiera 3 filósofos, ¿el problema sería de EMS? ¿Por qué?
Un problema es de exclusión mútua selectiva cuando, para acceder a un recurso, un proceso debe competir contra un grupo de procesos y no contra todos los procesos.

El problema de los cinco filósofos describe una mesa en donde hay 5 filósofos y 5 cubiertos entre ellos. Para comer, un filósofo necesita acceder a 2 cubiertos, por lo que compite contra los filósofos que tenga al lado. Así, solo 2 filósofos de los 5 podrán comer al mismo tiempo, y sobra un cubierto. Es un problema de EMS ya que cada filósofo compite solo con los 2 filósofos que tiene al lado, no contra los 4 restantes. Si hubiera 3 filósofos el problema dejaría de ser EMS, ya que cada proceso competiría contra todos los demás procesos por el acceso a los cubiertos.

6- Describa la técnica de Passing the Baton. ¿Cuál es su utilidad en la resolución de problemas mediante semáforos?
¿En qué consiste la técnica de Passing the Condition y cuál es su utilidad en la resolución de problemas con
monitores? ¿Qué relación encuentra entre passing the condition y passing the baton?
La técnica de Passing the Baton consiste en utilizar un token que represente el "permiso de ejecución". De esta manera, cuando un proceso está en la SC es porque tiene ese token, asegurando la exclusión mutua, y al salir se lo entrega a un proceso que esté esperando para entrar a la SC. En caso de no haber ningún proceso esperando, lo libera para que el próximo proceso que llegue lo tome. Cuando utlizamos semáforos, podemos utilizar un vector de semáforos privados, en donde cada proceso se quede a la espera de entrar a la SC, y cuando el proceso tenga que pasar el baton le haga un V al semáforo del proceso que quiere liberar.
La técnica de Passing the Condition plantea el mismo mecanismo de transpaso de "permiso", pero en este caso utilizando una variable condición en donde los procesos quedarán esperando. En este caso, los monitores tienen una variable condición donde los procesos pueden quedar "dormidos", entonces no es necesario tener un vector de semáforos. Cuando un proceso termina su ejecución y desea "pasar el permiso", lo que hace es hacer un signal sobre la variable condición en donde esperan los procesos (si es que hay alguno esperando).
Ambas técnicas comparten el principio de tener un "permiso" que nos permita entrar en la SC, el cual se vaya transfiriendo entre procesos a medida que salen o entran a la SC. En ambos casos, el proceso saliente entrega el permiso a un proceso que esté esperando, o lo libera si no hay ningún proceso a la espera. La diferencia entre uno y otro es la implementación del mismo, ya que en passing the condition se puede hacer una cola de procesos en espera en la variable condición, en cambio en passing the baton se tiene que hacer una implementación un poco más compleja con arreglos de semáforos.

7- Caracterice aplicaciones de grano grueso y de grano fino. ¿Cuál es la relación con la granularidad de procesos o
aplicaciones?
Las aplicaciones de grano grueso son aplicaciones que tienen una gran cantidad de cómputo y relativamente poca cantidad de comunicación, suele ser aplicaciones en segundo plano. Las aplicaciones de grano fino son aplicaciones que poseen una mayor cantidad de comunicación en comparación con la cantidad de procesamiento. La granularidad está dada por la relación entre el cómputo y la comunicación, una granularidad baja indica un proceso de grano grueso, y una granularidad alta indica un proceso de grano fino.

8- ¿En qué consiste la comunicación guardada y cuál es su utilidad? Describa cómo es la ejecución de sentencias de
alternativa e iteración que contienen comunicaciones guardadas.
La comunicación guardada es una técnica que nos permite utilizar el no determinismo a la hora de elegir una comunicación.
La comunicación guardada consiste en sentencias del tipo B;C --> S, en donde B es una condición booleana, C una comunicación y S es un conjunto de sentencias. Este conjunto de sentencias se ejecutan cuando el boolean es true y la comunicación se puede realizar.
Esas sentencias se puede utilizar con if y do, en estos casos se elige de forma no determinística entre las guardas verdaderas, es decir, aquellas líneas que tengan la condición booleana en verdadero y la comunicación lista para realizar.
En caso de que no haya ninguna guarda verdadera, cuando todas las condiciones booleanas son falsas, se sale de la estructura de control.
En caso de que la condición booleana sea verdadera, pero no se pueda realizar la comunicación, se queda en un bloqueo, esperando que algún par B;C esté listo para ejecutarse.
Si está dentro de un if, se sale de la estructura cuando todas las condiciones booleanas son falsas o cuando se ejecuta una sentencia. En el caso del do, solo lo hace cuando todas las condiciones booleanas son falsas.

3- Defina acciones atómicas condicionales e incondicionales. Ejemplifique.
Una acción atómica realiza una transformación de estados indivisible, es decir, mientras se ejecuta ningún estado intermedio es visible por otros procesos. Si la acción atómica es condicional, representa una sentencia await con una condición booleana, y hasta que no sea true no se puede ejecutar. Si el boolean es false, solo puede volverse true como resultado de las acciones de otros procesos. Si la acción atómica es incondicional, significa que no hay ninguna condición que lo

demora, por lo que puede ejecutarse inmediatamente.

5- ¿El problema de lectores y escritores es de EMS? ¿Por qué? Si sólo pudiera haber un lector accediendo a la Base de Datos, ¿el problema sería de EMS? ¿Por qué?
Si. Desde el punto de vista del escritor, este compite contra todos los demás procesos, por lo que no parecería ser EMS, pero los lectores solo compiten contra los escritores, por lo que ahí se daría la EMS.
Si solo podría haber un lector accediendo a la base, sería el mismo caso que se da con los escritores, en donde compite contra todos los demás procesos por el recurso, por lo que sería EMS.

3- Defina política de scheduling. Relacione con fairness.
Una política de scheduling define cual será la próxima acción atómica elegible que se va a ejecutar.
La acción atómica elegible de un proceso es la próxima acción atómica que se va a ejecutar, y la política de scheduling elige entre las acciones atómicas elegibles de todos los procesos.
El fairness intenta garantizar la continuidad y terminación de todos los procesos.
Según la política de scheduling, se pueden dar 3 tipos de fairness:

> Fairness Incondicional: una política de scheduling es incondicionalmente fair si toda acción atómica incondicional que se vuelve elegible es eventualmente ejecutada.
>
> Fairness Débil: una política de scheduling es débilmente fair si es incondicionalmente fair y si toda acción atómica condicional que se vuelve elegible es eventualmente ejecutada, asumiendo que su condición se vuelve true y permanece en ese estado hasta que es vista por el proceso que ejecuta la acción atómica condicional. Esto no asegura que cualquier sentencia await elegible se ejecuta eventualmente, ya que su guarda podría cambiar de valor mientras un proceso está demorado.
>
> Fairness Fuerte: una política de scheduling es fuertemente fair si es incondicionalmente fair y si toda acción atómica condicional que se vuelve elegible es eventualmente ejecutada pues su guarda se convierte en true con infinita frecuencia. Por lo general, no es simple tener una política que sea práctica y fuertemente fair.

5- Explicar en qué consiste el problema de la sección crítica, y cuáles son las 4 propiedades que se deben cumplir para resolverlo.
El problema de la sección crítica nos plantea que tenemos una o varias sentencias que no son atómicas y referencian a variables utilizadas por otros procesos. Por lo tanto necesitamos que se cumplan una serie de propiedades para asegurar un funcionamiento correcto:

- Exclusión mutua: A lo sumo un proceso está en su SC
- Ausencia de Deadlock (Livelock): si 2 o más procesos tratan de entrar a sus SC, al menos uno tendrá éxito.
- Ausencia de Demora Innecesaria: si un proceso trata de entrar a su SC y los otros están en sus SNC o terminaron, el primero no está impedido de entrar a su SC.
- Eventual Entrada: un proceso que intenta entrar a su SC tiene posibilidades de hacerlo y eventualmente lo hará.

Parte B

9- Dado el siguiente programa concurrente, indique cuál es la respuesta correcta (justifique)

```
int a = 1, b = 0;
co ⟨await (b = 1) a = 0 ⟩ // while (a = 1) { b = 1; b = 0; } oc
```

a) Siempre termina
b) Nunca termina
c) Puede terminar o no

10- Suponga los siguientes programas concurrentes. Asuma que a es un arreglo bidimensional, EOS es un valor especial que indica el fin de la secuencia de mensajes, y que los procesos son iniciados desde el programa principal.

P1

```
chan canal (double)
process Genera {
  int fila, col; double sum;
  for [fila= 1 to 10000]
    for [col = 1 to 10000]
      send canal (a(fila,col));
  send canal (EOS)   }
```

```
process Acumula {
  double valor, sumT;
  sumT=0;
  receive canal (valor);
  while valor<>EOS {
    sumT = sumT + valor
    receive canal (valor); }
  printf (sumT);
}
```

P2

```
chan canal (double)
process Genera {
  int fila, col; double sum;
  for [fila= 1 to 10000] {
    sum=0;
    for [col = 1 to 10000]
        sum=sum+a(fila,col);
    send canal (sum);    }
  send canal (EOS)   }
```

```
process Acumula {
  double valor, sumT;
  sumT=0;
  receive canal (valor);
  while valor<>EOS {
    sumT = sumT + valor;
    receive canal (valor); }
  printf (sumT);
}
```

a) Qué hacen los programas?
b) Analice desde el punto de vista del número de mensajes.
c) Analice desde el punto de vista de la granularidad de los procesos.
d) Cuál de los programas le parece más adecuado para ejecutar sobre una arquitectura de grano grueso de tipo cluster de PCs? Justifique.

9) El programa puede terminar o no.
El programa termina cuando el segundo proceso ejecuta b=1 y antes de ejecutar b=0 se realiza el await del primer proceso, que pondría a en 0 y finalizaría la ejecución del mismo. La ejecución continúa con b=0, a lo que la condición del while va a dar falsa y el segundo proceso va a terminar.
El programa puede no terminar cuando el primer proceso nunca llega a leer el valor de b entre que el segundo proceso haga b=1 y b=0, si el primer proceso siempre lee b=0 no va a terminar nunca.

10)
a) Los programas imprimen la suma de todos los valores de una matríz.
b) El primer programa envía más mensajes, ya que hace un envío por cada celda de la matríz, en este caso 10000^2 mensajes. El segundo programa envía menos, ya que hace un envío por cada fila, en este caso son 10000 mensajes.
c) El primer programa tiene una mayor granularidad ya que tiene un mayor grado de comunicación, a cada valor que lee lo envía. El segundo programa tiene una menor granularidad ya que tiene un mayor grado de cómputo, ya que va sumando los valores de cada fila y luego de eso recién hace la comunicación.
d) El segundo programa es más adecuado para ejecutar sobre una arquitectura de grano grueso de tipo cluster de PCs, ya que tiene una mayor cantidad de cómputo, acorde con lo que ofrece la arquitectura en cuestión, y tiene un menor grado de comunicación.

9- Dado el siguiente programa concurrente con variables compartidas:
x = 2; y = 5; z = 8;
co x = y - z // y = y * 3x // z = z * 2 oc
c.1) Cuáles de las asignaciones dentro del co cumplen con la propiedad de "A lo sumo una vez". Justifique.
c.2) Indique los resultados posibles de la ejecución. Justifique.
Nota 1: las instrucciones NO SON atómicas.
Nota 2: no es necesario que liste TODOS los resultados, pero sí todos los casos que resulten significativos.

10- Dado el siguiente bloque de código, indique para cada uno de los ítems si son equivalentes o no. Justificar cada caso (de ser necesario dar ejemplos).

Segmento 1	Segmento 2

…
int cant=1000;
While (true)
{ IF (cant > 15); datos?(cant) → *Sentencias1*
□ (cant < 5); datos?(cant) →*Sentencias2*
□ (**INCOGNITA**); datos?(cant) → *Sentencias3*
END IF
} …
	…
	int cant=1000;
	DO (cant > 15); datos?(cant) → *Sentencias1*
	□ (cant < 5); datos?(cant) → *Sentencias2*
	□ (**INCOGNITA**); datos?(cant) → *Sentencias3*
	END DO
	…

a) ***INCOGNITA*** equivale a: ***(cant = 5) or (cant = 15)***
b) ***INCOGNITA*** equivale a: ***(cant > 0)***
c) ***INCOGNITA*** equivale a: ***((cant >= 2) and (cant <= 20))***
d) ***INCOGNITA*** equivale a: ***((cant > 5) and (cant <= 15))***
e) ***INCOGNITA*** equivale a: ***((cant > 5) and (cant < 15))***

9)
c1) La asignación que cumple con la propiedad a lo sumo una vez es la z = z * 2, ya que no tiene referencias críticas, porque z no es modificada por otro proceso, y z puede ser leída por otro proceso.
c2) x = -3 ; y = -45 ; z = 16
x = -11 ; y = -165 ; z = 16
x = 14 ; y = 30 ; z = 16
x = 22 ; y = 30 ; z = 16

10)
a) No es equivalente
b) Es equivalente
c) Es equivalente
d) No es equivalente
e) No es equivalente

Parte C

11- Compare los algoritmos para resolver el problema de la Sección Crítica (spin locks, tie breaker, ticket, bakery), marcando ventajas y desventajas de cada uno. (NO implemente).
Spin lock:

-Ventajas: simple de implementar
-Desventajas: utiliza instrucciones especiales que pueden no estar en todos los procesadores. Queda iterando todo el tiempo esperando a que la variable cambie su valor (busy waiting). No controla el orden de los procesos demorados, por eso requiere scheduling fuertemente fair (aunque el débil es aceptable).

Tie Breaker:

-Ventajas: no utiliza intrucciones especiales y requiere solo un scheduling débilmente fair.
-Desventajas: es un algoritmo cuya generalización resulta compleja y costosa en tiempo.

Ticket:

-Ventajas: es fácil de implementar con varios procesos. (poneeeeele)
-Desventajas: los valores del turno y próximo se deben reiniciar cada cierto tiempo. Si no existe la instrucción especial FA la solución puede no ser fair.

Bakery:

-Ventajas: es fair y no requiere instrucciones especiales ni un contador global para próximo.

-Desventajas: es más complejo.

12-
a) Defina las métricas de speedup y eficiencia. ¿Cuál es el significado de cada una de ellas (qué miden)? ¿Cuál es el rango de valores posibles de cada uno? Ejemplifique.
El speedup es una métrica que mide cuanto más rápida es mi solución paralela con respecto a la mejor soiución serial. El rango de valores va de 0 al speedup óptimo (el cual es el mismo que el número de procesadores que utilizo, si tengo 8 procesadores, mi speedup óptimo es 8). En la realidad el valor suele ser menor al óptimo, pero puede ser mayor que el óptimo incluso en algunos casos (speedup superlineal).

La eficiencia es el cociente entre el speedup obtenido y el speedup óptimo, indicando cual es la fracción del tiempo en la que los procesadores son útiles para el cómputo, para determinar que "desperdicio" de cómputo hay en ese algoritmo. La eficiencia va de 0 a 1, siendo 1 la representación de que todos los procesadores están siendo utilizados todo el tiempo (lo óptimo).

b) Qué se entiende por escalabilidad de un sistema paralelo?
La escabilidad determina la capacidad que tiene un sistema de mantener la eficiencia constante cuando crece la cantidad de procesadores y los datos a procesar. Que una solución sea escalable determina que, si tiene una buena performance con pocos datos, también la va a tener con muchos procesadores y muchos datos.

c) Suponga que la solución a un problema es paralelizada sobre p procesadores de dos maneras diferentes. En un caso, el speedup (S) está regido por la función S=p-7 y en el otro por la función S=p/3. ¿Cuál de las dos soluciones se comportará más eficientemente al crecer la cantidad de procesadores? Justifique claramente.
La solución más eficiente es p-7, ya que al crecer la cantidad de procesadores va a decrecer menos el speedup que con p/3. Por ejemplo, si tuvieramos 15 procesadores, la solución S=p-7 daría un speedup de 8, en cambio la solución p/3 daría un speedup de 5, y la diferencia aumenta a medida que aumentan los procesadores. Con 150 procesadores, S = p-7 da un speedup de 143, mientras que p/3 da un speedup de 50.

d) Suponga que el tiempo de ejecución de un algoritmo secuencial es de 10000 unidades de tiempo, de las cuales sólo el 95% corresponde a código paralelizable. ¿Cuál es el límite en la mejora que puede obtenerse paralelizando el algoritmo? Justifique
El límite de la mejora paralelizando el algoritmo va a ser de 501 unidades de tiempo. Como el 5% del código no es paralelizable, es un tiempo que no podemos reducir, es decir, solo las 9500 unidades de tiempo se pueden mejorar. Si tuvieramos 9500 procesadores, esas 9500 unidades de tiempo (en un speedup óptimo) pasarían a valer 1, y el algoritmo total tardaría 1 + 500 unidades de tiempo.

13-
Defina los paradigmas de interacción entre procesos distribuidos heartbeat, servidores replicados, prueba-eco, master-worker y token passing. Marque ventajas y desventajas cuando se utiliza comunicación por mensajes sincrónicos o asincrónicos.
Heartbeat: en este paradigma los procesos intercambian información, con mecanismos tipo send/receive, de forma periódica, en cada paso un proceso comparte la información con sus vecinos, y en el próximo paso utiliza esta información nueva para procesar. En caso de tener una red, los mensajes van viajando desde un nodo, expandiendose por la red, y llega un punto en donde el flujo de información "se contrae", volviendo al nodo origen.

Servidores replicados: el paradigma de servidores replicados plantea replicar servidores para que cada uno de ellos maneje una instancia de un recurso compartido, en lugar de tener varias instancias en un solo servidor. Permite incrementar la accesibilidad a los datos o servicios, donde cada servidor interactúa con los demás para darle al cliente la sensación de que es un único servidor.

Prueba-eco: es un paradigma particularmente útil para recorrer estructuras de tipo grafo o árbol, que consiste en que un nodo envía a sus sucesores un mensaje "probe", y espera el retorno de la información con un mensaje "echo". A su vez, antes de hacer el echo, el nodo receptor del "probe" original, envía un "probe" a sus sucesores, y así sigue la cadena hasta recorrer toda la estructura.

Master-worker: es un paradigma que se caracteriza por su fácil escalabilidad y su equilibrio en la

carga de trabajo en los workers. Consiste en tener una bolsa o "bag of task", la cual es una estructura en la que se van apilando tareas. A su vez, están en ejecución un conjunto de procesos, los cuales de manera independiente van tomando tareas de esta bolsa y resolviendolas, también puede apilar nuevas tareas de ser necesario. En la implementación con pasaje de mensajes, la "bolsa" está representada por un proceso que acumula las tareas y se va comunicando con los workers en un esquema cliente-servidor.

Token passing: es un paradigma que se suele utilizar en arquitecturas distribuídas. Consiste en el viaje de un tipo especial de mensaje "token". Este token suele tener dos utilidades, una es ir almacenando información a medida que se desplaza por la arquitectura, la otra es utilizarlo como un mecanismo de control, por ejemplo, si tengo un recurso compartido logicamente entre varios procesos, quien tenga el token será el proceso que tiene el "poder de ejecución" sobre ese recurso. También permite la toma de decisiones distribuída.

En ambos casos:
No se necesita hacer polling para recibir un mensaje.
El acceso a los canales es atómico, por lo que no es necesario agregar sincronización.

PMA:

-Ventajas: puede utilizar los 3 tipos de canales (link, mailboxes o input port). Los canales son una cola, por lo que se pueden acumular mensajes y respetan un orden.
-Desventajas: como el send no es bloqueante, en algunos casos se debe agregar una sincronización a los procesos.

PMS:

-Ventajas: la condición bloqueante de la instrucción send se puede utilizar como sincronización. Puede simular PMA con un proceso buffer.
-Desventajas: la instrucción send es bloqueante (aunque no siempre es una desventaja, en algunos casos se le encuentra una utilidad, pero en términos generales suele degradar la performance). Los canales son solo del tipo link. Solo puede acumularse un mensaje por canal. Mayor posibilidad de deadlock.

11- a) Describa brevemente en qué consisten los mecanismos de RPC y Rendezvous. ¿Para qué tipo de problemas son más adecuados?
RPC y Rendezvous son técnicas de comunicación y sincronización entre procesos que suponen un canal bidireccional, ideales para programar aplicaciones C/S. Tienen una interfaz "tipo monitor", en donde exportan operaciones con llamadas externas con mensajes sincrónicos.
RPC en particular crea un nuevo proceso para atender cada llamada a un procedimiento, por lo que puede atender en simultaneo varias llamadas al procedimiento. RPC solo hace la comunicación, la sincronización debe ser agregada mediante los métodos que ya conocemos, ya que el único trabajo del servidor que realiza el procedimiento es actuar como si fuera parte del proceso original.
Rendezvous por su parte, desarrolla la comunicación y la sincronización. A diferencia de RPC, el llamado a un procedimiento es atendido por un proceso ya existente, el cual está en espera constantemente hasta que alguien hace un llamado. Por esta razón, solo se puede atender de a una llamada al procedimiento a la vez.

b) ¿Por qué es necesario proveer sincronización dentro de los módulos en RPC? ¿Cómo puede realizarse esta sincronización?
RPC no ofrece sincronización por sí mismo, la única sincronización que hay es la del proceso llamador con la del proceso servidor que realiza la tarea, pero es necesaria programar la sincronización del proceso servidor con la de los demás procesos que están en el módulo, tanto para sincronizar las interacciones como para acceder a los recursos compartidos, ya que comparten el mismo espacio de memoria.

c) Qué elementos de la forma general de rendezvous no se encuentran en el lenguaje ADA?
Las operaciones in ni no son soportadas de forma completa. Lo implementa mediante una sentencia select que es más reducida en capacidades, ya que no puede referenciar argumentos a operaciones o contener expresiones de scheduling.

Parte D

14- Sea la siguiente solución al problema del producto de matrices de nxn con P procesos en paralelo con variables compartidas.

```
process worker [w = 1 to P] { # strips en paralelo (p strips de n/P filas)
  int first = (w-1) * n/P + 1       # Primera fila del strip
  int last = first + n/P - 1;       # Ultima fila del strip
  for [i = first to last] {
     for [j = 1 to n] {
          c[i,j] = 0.0;
          for [k = 1 to n]
            c[i,j] = c[i,j] + a[i,k]*b[k,j];
        }
     }
}
```

Suponga n=256 y cada procesador capaz de ejecutar un proceso.
a) Calcular cuántas asignaciones, sumas y productos se hacen secuencialmente (caso en que P=1), y cuántas se realizan en cada procesador en la solución paralela con P=8.
b) Si los procesadores P1 a P7 son idénticos, con tiempos de asignación 2, de suma 4 y de producto 6, y si el procesador P8 es 2 veces más lento, calcule cuánto tarda el proceso total concurrente
c) ¿Cuál es el valor del speedup?
d) ¿Cómo modificaría el código para lograr un mejor speedup?
NOTA: para realizar los cálculos no tenga en cuenta las operaciones de asignación e incremento correspondientes a las sentencias *for*.

a)
Secuencial:
//Acá las cuentas se hacen por 256 (como si fueran 32 filas x 8 procesadores), ya que la carga no se divide entre los
//procesadores
Asignaciones = 256^3 + 256^2 = 16.842.752
Sumas = 256^3 = 16.777.216
Productos = 256^3 = 16.777.216

Paralelo:
Asignaciones = 256^2 * 32 + 256*32 = 2.105.344
Sumas = 256^2*32 = 2.097.152
Productos = 256^2*32 = 2.097.152

b)
P1 a P7 = 2.105.344 * 2 + 2.097.152 * 4 + 2.097.152 * 6 = 25.182.208
P8 = 25.182.208 * 2 = 50.364.416

c)
Ts = 16.842.752 * 2 + 16.777.216 * 4 + 16.777.216 * 6 = 201.457.664
Tp = 50.364.416
Speedup = 201.457.664 / 50.364.416 = 4
El valor del speedup es de 4

d) Para lograr un mejor speedup hay que repartir la carga de forma no homogenea, es decir, a P8 se le asigna una carga menor y a los procesadores P1 a P7 una carga un poco mayor, para intentar que todos terminen al mismo tiempo y la eficiencia sea la máxima.