
Paralelismo - 23GIIN

Actividad 2 - Portafolio

Gagliardo Miguel Angel

26 de Noviembre de 2023

1. Usar la herramienta GPROF para determinar el perfil de ejecución de:

- **average.c**

- **addMatrix.c**

Para cada ejercicio indique cual es la función que mas tiempo consume y qué estrategia de paralelización (por datos, por función, etc.) usaría para reducir el tiempo total de ejecución.

average.c

```
Each sample counts as 0.01 seconds.
%   cumulative   self           self         total
time  seconds    seconds    calls   ms/call  ms/call  name
61.54      0.48      0.48             1    300.00   300.00   main
38.46      0.78      0.30             1    300.00   300.00  func_average
```

```
granularity: each sample hit covers 4 byte(s) for 1.28% of 0.78 seconds

index % time    self  children    called    name
-----
[1]   100.0    0.48    0.30             1/1    <spontaneous>
      0.30    0.00             1/1    main [1]
      -----
      0.30    0.00             1/1    func_average [2]
[2]   38.5    0.30    0.00             1    main [1]
      0.30    0.00             1    func_average [2]
      -----
```

- Como se puede ver en este caso, la función que consume la mayor cantidad de tiempo es **main** con un **61%** del tiempo total de ejecución.
- La estrategia de paralelización que en este caso usaría para reducir el tiempo total de ejecución es la de descomposición **por dominio**. Dado que la función **main()** primero popula el vector **num** con datos y luego llama a **func_average** que realiza una suma de todos los datos en ese vector (sobre una variable nueva sum) y luego la divide por el largo del vector (N).

- **Por dominio**, dado que podemos poblar el vector main de manera mucho mas rapida simplemente partiendo el dato de N en varias partes y con un puntero que indique a que altura del mismo estamos, asignarle datos a dicha altura solo utilizando un puntero incremental y el dato en la posicion deseada. Esta tarea puede ser paralelizable y ejecutada de manera concurrente dado que, mas alla de la posicion en el vector, cada dato es independiente del anterior.
- **Un ejemplo** de esta descomposicion seria sumar la serie de números en paralelo. Dicha descomposición requiere dividir el vector N en partes iguales y asignar cada parte a un procesador. El cálculo se produce en dos fases. Suponiendo que tenemos 4 procesadores y **en primer lugar**, el trabajo se dividiria a partes iguales entre los mismos. En este caso, cada procesador suma una cantidad similar de números en la matriz. El procesador numero 1 al que llamaremos **maestro** suma todos los elementos que comienzan en el índice 0 y terminan en el índice X, el segundo procesador suma todos los elementos que comienzan en el índice X+1 y terminan en el índice Y, y el tercer procesador suma todos los elementos que comienzan en el índice Y+1 y terminan en el índice Z, y el ultimo procesador (numero 4) suma todos los elementos que comienzan en el índice Z+1 y terminan en el índice N (largo del vector). **La segunda fase**, ocurre luego de que todos los procesadores hayan terminado con su fase, aqui **el procesador maestro agrega todas las sumas** asignadas a las distintas funciones que han ejecutado los otros procesadores, para obtener un total “final” y ejecuta la division sum/N como se muestra en este grafico.
- Este tipo de paralelizacion es secuencial dado que al final necesitaremos esperar a que todos los procesadores realicen sus sumas para poder, finalmente, realizar el agregado.



(b) Comunicación descentralizada, pero secuencial

addMatrix.c

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ms/call	total ms/call	name
66.67	0.02	0.02	1	20.00	20.00	func_sum
33.33	0.03	0.01	1	10.00	10.00	inicial

index	% time	self	children	called	name
					<spontaneous>
[1]	100.0	0.00	0.03		main [1]
		0.02	0.00	1/1	func_sum [2]
		0.01	0.00	1/1	inicial [3]

		0.02	0.00	1/1	main [1]
[2]	66.7	0.02	0.00	1	func_sum [2]

		0.01	0.00	1/1	main [1]
[3]	33.3	0.01	0.00	1	inicial [3]

- Como se puede ver en este caso, la función **func_sum** se lleva la mayor parte del tiempo de ejecución, siendo 66.7% del total.
- La estrategia de paralelización que en este caso usaría para reducir el tiempo total de ejecución es la de descomposición **por dominio**, dado que no tenemos una gran cantidad de datos (matrices cuadradas NxN con N=2000), pero sobre todo porque se realiza una gran cantidad de computo.
- De la misma manera que en el ejercicio anterior, podemos asignar las diversas sumatorias i+j a los procesadores. Luego el procesador maestro repartiría (nuevamente) los datos a los distintos procesadores a medida que se vayan liberando a fin de que cada uno de ellos ubique en la matriz sum[i][j] el dato producto de la suma que ha obtenido con la posición correspondiente. Cabe destacar que en este caso todas las tareas pueden ser asincrónicas dado que los datos son siempre independientes, en tanto y en cuanto se tengan en cuenta los punteros i, j de la matriz que se va a llenar.

2. Plantee el diseño paralelo usando la metodología PCAM vista en clase de los dos enunciados mostrados más adelante (pueden utilizar gráficos que ilustren cada paso de PCAM). Se trata de plantear el diseño según crean conveniente. No se trata de implementarlo ni de conseguir una implementación en Internet.

a) Suponga que tiene una lista de N números enteros (N muy grande), que llamaremos ListaN y otra lista de M números primos, donde $M \ll N$ (M es mucho menor que N), que llamaremos ListaPrimos. Se desea encontrar la lista de números en ListaN que sean múltiplos de todos los números primos de ListaPrimos.

El ejercicio claramente propone que como primer inconveniente tenemos una lista de números muy grande **ListaN**, **pero** cada operación de encontrar cada número que sea múltiplo de todos los de la lista **ListaPrimos** es **independiente** de la otra. Lo cual quiere decir que no solo no tengo dependencia entre datos, si no que puedo paralelizar la ejecución misma.

Por tanto, el planteo propuesto es:

- **Particionamiento (P):** La idea es realizar una descomposición **por datos** a través de **tareas paralelas** dado que, como mencionaba antes, la búsqueda de cada múltiplo es independiente y cada procesador retornará un dato independiente de los demás, todos trabajarán con un dato en común (ListaPrimos) **pero la tarea es idéntica: Es este número X múltiplo de todos los números de ListaPrimos?**
- **Comunicación (C):** La propuesta es que sea de tipo **Local, Estructurada, Estática y Asíncrona**. **Local** porque no hay una gran necesidad de vincular las tareas, porque cada una no necesita excesiva comunicación con otras y porque tampoco tienen por qué interferir (bloquear) otras comunicaciones. **Estructurada** porque que todos los nodos ejecutarían copias idénticas de una tarea (búsqueda del múltiplo en ListaPrimos) pero utilizando el mismo vector (ListaN) el cual **no debe ser replicado** para evitar un overflow. Esto ayudaría a la escalabilidad del programa, o sea tener un mayor número de procesadores que beneficien el cálculo. **Estática** porque la identidad de los modelos de comunicación no cambiara con el tiempo y los tiempos de cómputo no se espera sean variables, dado que las listas N y M son **estáticas y por lo tanto sus tareas asociadas también**, y finalmente **asíncrona** porque las tareas (nuevamente, la búsqueda de cada número de ListaN que es múltiplo de todos los números primos de ListaPrimos) es independiente, y no necesita ser ejecutada de manera coordinada ni

bloqueante. Lo único que importa es que nuestro servicio devuelva, finalmente, una lista de números que no necesitan ser calculados de manera dependiente el uno del otro.

- **Aglomeración (A):** En cuanto a aglomeración, dado que tendremos un vector grande ListaN, podemos aglomerar subdividiéndolo en porciones más pequeñas. Así, por ejemplo, tendremos vectores ListaN más chicos que podremos fácilmente asignar a cada procesador.
- **Mapping (M):** Como antes se menciona, la tarea de búsqueda del número que sea múltiplo de todos los números de ListaPrimos puede ser ejecutada de manera concurrente, por tanto dichas tareas se deben asignar a procesadores diferentes a través de un **mapeo cíclico**.

b) Un popular juego de crucigramas consiste en encontrar palabras en una matriz cuadrada, en cualquiera de las posiciones verticales o horizontales. Suponga que tiene una matriz cuadrada de N (muy grande) de letras y una lista de M palabras que debe buscar en la matriz de caracteres. No todas las M palabras aparecen en la matriz. Se desea contar cuántas de las M palabras aparecen en la matriz. Las palabras pueden estar en horizontal o vertical.

En este caso vemos como el ejercicio propone nuevamente tareas **independientes**, dado que debemos encontrar una lista de palabras **M** (cada palabra no depende de la anterior) en una matriz cuadrada $N \times N$ con **N muy grande** en, idealmente, una sola pasada para minimizar el procesamiento. Donde, suponiendo recorreremos la matriz de manera lineal (ejemplo línea por línea, o columna por columna), al encontrar la primera letra que corresponde a una palabra de la lista **M**, buscar sus vecinos (en 4 direcciones – dado que pueden estar en dirección vertical u horizontal) y verificar si las letras consecuentes corresponden a todas las otras letras de dicha palabra de la lista **M**. Si, finalmente, se verifica dicha palabra se aumenta un contador en +1 y, finalmente, ese es el dato a retornar.

Por tanto, el planteo propuesto es:

- **Particionamiento (P):** La idea es realizar una descomposición **por datos** a través de **tareas paralelas** dado que, como mencionaba antes, la búsqueda de cada palabra de la lista **M** es independiente de las demás (una palabra no depende de la otra) y cada procesador retornará un dato independiente de los demás (Verdadero o Falso, según corresponda) aunque todos trabajaran con un dato en común (matriz $N \times N$) **pero la tarea es idéntica: Se encuentra esta palabra de la lista M en la matriz $N \times N$?**

- **Comunicacion (C):** La propuesta es que sea de tipo **Local, Estructurada, Estatica y Asincrona**. **Local** dado que no necesitamos comunicacion entre las tareas, porque buscar una palabra en la matriz no depende de las anteriores, o sea no necesita excesiva comunicacion con otras y porque tampoco tienen por que interferir (bloquear) otras comunicaciones. **Estructurada** porque que todos los nodos ejecutarían copias idénticas de un programa (busqueda de una palabra en una Matriz) mas utilizando la misma matriz (N) la cual **no debe ser replicada** para evitar un overflow. Esto ayudaría a la escalabilidad del programa, o sea tener un mayor numero de procesadores que beneficien el calculo. **Estatica** porque la identidad de los modelos de comunicacion no cambiara con el tiempo y los tiempos de computo no se espera sean variables, dado que tanto la matriz como las palabras son listas **estaticas y por lo tanto sus tareas asociadas tambien**, y finalmente **asincrona** porque las tareas (nuevamente, la busqueda de cada palabra en la matriz cuadrada) es independiente, y no necesita ser ejecutada de manera coordinada ni bloqueante. Lo unico que importa es que nuestro servicio devuelva, finalmente, un numero que corresponde a la cantidad de palabras que se encuentran en la matriz.
- **Aglomeracion (A):** En cuanto a aglomeracion, dado que tendremos una matriz cuadrada grande podemos aglomerar subdividiendo M (cantidad de palabras) y así minimizar la cantidad de tareas por procesador. Así, por ejemplo, tendremos vectores M mas chicos que podremos facilmente asignar a cada procesador.
- **Mapping (M):** Como antes se menciona, la tarea de busqueda de la palabra que se encuentre en la matriz puede y debe ser ejecutada de manera concurrente, por tanto dichas tareas se deben asignar a procesadores diferentes a traves de un **mapeo ciclico**.