**Is parallelism for You?**

Reglas de oro para científicos en la computación e ingenieros.

En este artículo, se comienza por realizar una breve introducción, en donde se describe la programación paralela a grandes rasgos, la cual se basa en la idea de que múltiples CPUs pueden solucionar problemas más complejos y de una forma más rápida, sin embrago, muchas veces resulta bastante costosa, llegando hasta el punto en que realmente funcionan más lenta que la programación en serie. Es por esto que, este artículo te proporciona una pequeña guía para saber si es conveniente o no que realices un programa utilizando programación paralela. La información es más que nada empírica, de científicos e ingenieros en computación.

Primero, se describen unos prerrequisitos. Estos se describen a través de un diagrama, en el cual se pueden resumir tres rubros principales:

* *La frecuencia de uso:* Hace referencia a que tan utilizado será el programa antes de que sea modificado. Si tendrá un gran uso antes de realizarle cambios, entonces el programa es buen candidato para paralelizar. En caso contrario, podría no ser lo más eficiente
* *El tiempo de ejecución:* En caso de que el programa tenga un tiempo de ejecución bastante alto, llegando hasta el orden de días o que se necesiten bastantes horas para poder terminar, entonces también es buen candidato para paralelizarse.
* *La complejidad:* Cuando la complejidad o velocidad se aumenta considerablemente.

Con tener al menos una de estas, entonces sería bastante probable que el programa tenga que paralelizarse, no obstante, aún ese necesario tener en consideración las reglas de oro para dar el paso definitivo.

Antes de describir las primeras cuatro reglas de oro, es necesario hablar de las diversas formas en como se puede modelar un problema haciendo uso de la programación paralela.

La primera hace referencia a cuando un problema se puede modelar con paralelismo perfecto, que es el más simple. En este caso, los cálculos de información son independientes uno de otro, por lo que, los datos se pueden procesar en distintos procesadores sin ningún problema. Este tipo de paralelismo permite lograr grandes mejoras de rendimiento.

Ahora, en caso de que los datos no sean completamente independientes, que se tenga una cierta relación en donde, primero se le haga un tipo de proceso a los datos y el resultado de esta se utilice en otro proceso, sucesivamente hasta n procesos, entonces, se tendría un paralelismo de tubería. El paralelismo de tubería empieza a presentar dificultades, como el hecho de distribuir de manera equitativa la carga de trabajo, para que los procesadores no realicen un proceso más rápido que otros, y se queden esperando información o tengan que almacenarla momentáneamente pues el proceso siguiente aún no está listo.

Por otro lado, si en una iteración, distintos procesos o cálculos se tiene que realizar a subconjuntos de datos utilizando diversos procesadores, entonces se tiene un paralelismo totalmente sincronizado, donde se realizan los cálculos o procesos de manera sincronizada a todos los datos. Lo importante a remarcar es que, los cálculos o procesos realizados dependen de los anteriores.

En este caso, se tiene el problema de que, si los subconjuntos de datos formados no son homogéneos, entonces se tendrán procesadores trabajando bastante, mientras que existan otros procesadores que no estén realizando nada, por lo que, se debe hacer un esfuerzo aún más grande en este tipo de paralelismo para logar un buen rendimiento.

Ahora, si la cantidad de datos va cambiando conforme el tiempo, entonces se tiene un paralelismo débilmente sincronizado, en donde los procesadores se tienen que comunicar para ir intercambiando información. Este es el tipo de paralelismo más difícil, pues ahora se tiene que distribuir la carga de trabajo dependiendo del tiempo y la cantidad de datos en ese momento de tiempo.

Es en este punto, se ofrecen cuatro reglas de oro, las cuales se resumen en que, si tu problema se puede modelar con paralelismo perfecto, entonces lo ideal podría ser paralelizar. Conforme se llegan a los diversos paralelismos enlistados anteriormente, se va dificultando la paralelización del problema, por lo que, podría no ser ideal paralelizar el problema, pues podría llegar a complicarse mucho y no ofrecer buenas mejoras en el rendimiento y/o complejidad.

Una vea definidos los tipos de paralelismos, se define la manera en como se deben distribuir y comunicar los procesadores para poder realizar este tipo de paralelismos.

En la arquitectura SIMD (single instruction, multiple data) a todos los procesadores se les da una operación a realizar con diferentes datos. Este tipo de arquitectura se ajusta bastante al paralelismo perfecto. La desventaja de esta arquitectura es que puede haber ocasiones en que se tenga que realizar una operación solo si se cumple una condición, por lo que, si en algunos procesadores no se cumple la condición, entonces estos no realizarán tarea alguna y estarán sin realizar nada. De igual forma, si todos los procesadores realizan la misma operación, entonces solo se tomará en cuenta el que termine primero.

En la arquitectura MIMD (multiple instruction, multiple data) los procesadores acceden a una memoria compartida, en donde realizan diversas instrucciones de una operación de datos almacenados en la memoria compartida. Aquí, se tiene que proteger algunas veces los datos para que, mientras un procesador está modificando algunos datos, otro procesador no pueda modificar esos datos. Este tipo de arquitectura se ajusta bastante bien al paralelismo de tubería o al paralelismo débilmente sincronizado.

En la arquitectura de memoria distribuida, cada procesador tiene su propia memoria y se comunica con otros procesadores mediante mensajes con datos. En esta arquitectura, se puede presentar livelocks (cuando un procesador se queda esperando por información) y deadlocks (cuando dos procesadores se bloquean al quedarse esperando información entre ambos). De igual forma, se tiene el problema de comunicación, ya que muchas veces puede ser tardada la comunicación entre procesadores. Este tipo de arquitectura se ajusta al paralelismo completamente independiente.

En última instancia, se tienen los SMP’s y SMP clusters, los cuales son un tipo combinación de las anteriores arquitecturas, ya que se tiene que dentro de un cluster (grupo) existen varios procesadores, por lo regular cuatro u ocho, que comparten una misma memoria, por lo que, se tienen varios clusters, todos con la misma cantidad de procesadores y su respectiva memoria compartida.

Estos multiprocesadores se ajustan también a el paralelismo de tubería.

Se describen otras cuatro reglas de oro, las cuales consisten en poder adaptar los tipos de paralelismo a una respectiva arquitectura sin mucha dificultad.

Otro aspecto importante a considerar, es el lenguaje de programación, en donde, se describe otra regla de oro, la cual dice que tú no escoges el lenguaje de programación, si no que el te escoge a ti, haciendo referencia a que muchas veces solo habrán algunos lenguajes de programación que se ajusten a la máquina donde se trabajará el algoritmo paralelizado y la arquitectura de paralelización que seguirás, por lo que solo podrás escoger entre algunos para trabajar, pudiendo llegar el caso en que no tengas conocimiento en ninguno de los lenguajes.

Para finalizar el artículo, se habla de establecer expectativas realistas al momento de tratar con programación paralela. Para poder lograrlo, primero se deben de haber seguido las reglas de oro anteriormente citadas y tener un programa en serie para tomarlo como base. Esto es importante, pues el hecho de tener un programa en serie como base nos permite verificar que, al paralelizarlo, producen los mismos resultados, además de que sirve para comparar que tan rápido es su versión paralela.

Para comparar la mejora en el rendimiento, se recomienda medir el tiempo que toman aquellas partes del código que se desean paralelizar y, en general, el tiempo total que tarda el código. Con esto, se aplica una fórmula para encontrar la cantidad de código potencial a paralelizar, dividiendo la cantidad de tiempo que toman las regiones a paralelizar entre la cantidad total de tiempo del programa. El resultado de esta fórmula se guarda en p. De igual forma, se nos ofrece otra fórmula para determinar el rendimiento teórico que podríamos obtener. La fórmula consiste en sacar el inverso de la suma de 1 – p y p entre el número total de procesadores a ocupar. Con este resultado podemos ir variando la cantidad de procesadores a ocupar e ir graficando dichos resultados, para conocer el número adecuado de procesadores a utilizar.

Es importante remarcar que, si bien obtuvimos un rendimiento teórico, este se ve bastante afectado por otros factores, como la latencia entre los procesadores, donde un mensaje tarda en llegar debido a la cantidad de información que transporta el mensaje, la conectividad entre procesadores, la velocidad de cada procesador, si un procesador llega a quedarse esperando a recibir un mensaje, entre otros factores. A partir de esto último, surgen las últimas reglas de oro.

Este artículo nos da una idea general de lo que es la programación paralela, pues muchas veces, al hablar de ello, se nos forma la idea de que el hecho de tener más procesadores nos permitirá un mayor rendimiento y que debería ser siempre la mejor opción, sin embargo, a través del artículo, podemos entender que puede resultar no tan eficiente paralelizar un código, ya que se deben considerar bastantes cuestiones, empezando por los prerrequisitos, como el hecho de cuanto tardaría en ejecutarse el programa, su uso, entre otras cuestiones, hasta llegar a dar aproximaciones de que tanto rendimiento esperamos, considerando factores como la latencia de los procesadores, su velocidad, etc.

La paralelización es una herramienta bastante poderosa, siempre y cuando se aplique sobre problemas que no sean tan complicados de paralelizar y después, escoger su arquitectura preferente, el lenguaje de programación adecuado, así como realizar mediciones de tiempo para determinar que tanta optimización teórica se podría obtener, para que finalmente, con base en toda esta información, se pueda tomar la decisión de paralelizar o no hacerlo.