Trabajo de Investigación de Algoritmos Paralelos

Redes Interconectadas

Memoria-compartida

El modelo de memoria compartida consta de múltiples CPUs, estos con diferentes núcleos pero con el mismo código; estos CPUs comparten un Bus de coneccion a memoria, que los comunica con la memoria (donde se almacenan los datos) y con los elementos de I/O.
Para que una red de memoria compartida pueda funcionar, se debe iniciar con un **thread** que sea padre de otros **threads**, que se busquen paralelizar en distintos CPUs; estos threads a su vez pueden ser padres de más threads que se busquen paralelizar. El thread padre esperará a que sus hijos terminen sus tareas y tras estos ejecutara un mecanismo de limpieza.

Threads dinamicos y estaticos

Los threads **dinámicos** son aquellos que a través de un thread maestro reciben tareas (usualmente tras ser recibida por red), es en este momento donde son creados y luego se espera su culminación.
Los threads **estáticos** en cambio, son lanzados todos para una misma tarea por un thread maestro y, tras su culminación y posterior limpieza, el thread maestro culmina con ellos. Este paradigma es considerado menos eficiente.

Problemas

No determinístico

Parte de la primicia que de una entrada puedan salir varias salidas que no necesariamente sean las correctas. Se puede dar **Race Conditions** cuando un thread compite con otro para ver quién acaba primero, logrando que estos no tengan una correcta salida como resultado.
Se ve entonces una **sección crítica** donde un thread debe acceder antes que los demas o un grupo de los demás; es necesaria una **exclusión mutua** entre threads donde se acceda a los recursos de manera ordenada.

Soluciones

Locks que bloqueen la sección crítica haciendo que los demás threads no accedan o no avancen hasta su culminación.
Semáforos que funcionan de manera similar a los locks pero mejores en ciertos escenarios.
Monitores, similares a los looks que solo pueden ser modificados por un thread a la vez.

Más problemas

Muchas de estas soluciones pueden hacer que el desempeño del algoritmo paralelo se vea reducido o anulado; estas son algunos de los problemas que se pueden ocasionar por estas:

Busy Waiting donde se espera sin hacer nada hasta que se culmine la seccion critica, como su nombre lo dice se consume recursos mientras no se hace nada.

Serialización, nuestro algoritmo tiene un comportamiento casi lineal por no estar paralelizando realmente.

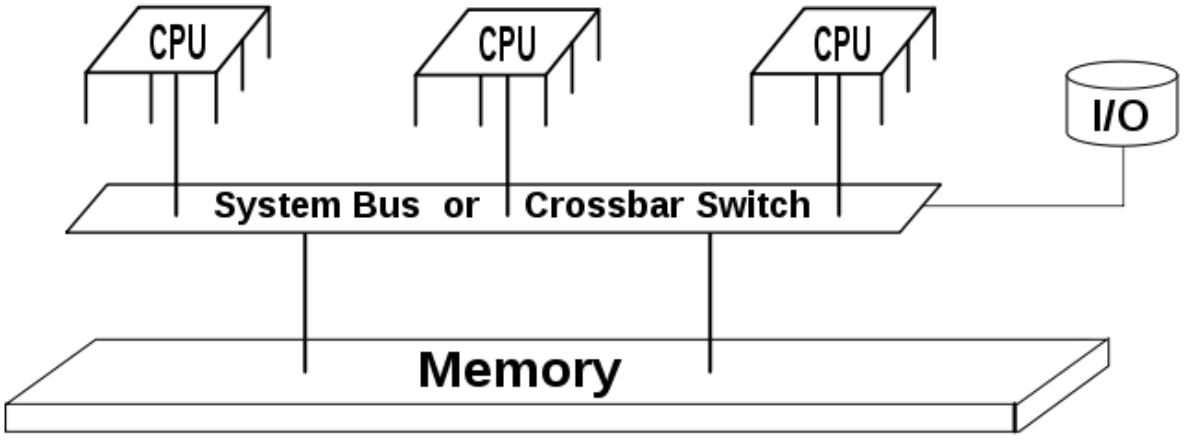
Recomendaciones

Es necesario optimizar el comportamiento durante el paralelismo es por eso que se recomienda utilizar funciones propias de la arquitectura para garantizar un comportamiento adecuado.

Resumen y datos adicionales

En una interconección de memoria compartida se asignan threads por medio de un thread padre, estos pueden tener un paradigma estatico o dinamico; para lograr mejor desempeño se recomienda usar funciones propias del modelo en el que se está trabajando; además, se trata de minimizar la serialización y la busy waiting a través de optimización del algoritmo.

Se recuerda que este modelo trabaja con thread padre y threads hijos en una sola computadora lo cual podría presentar un problema cuando se quiere hacer cosas más grandes.



Distributed-memory

Distributed-memory utiliza una red para conectar varias computadoras donde se transmite procesos a cada una de estas.

Message-passing

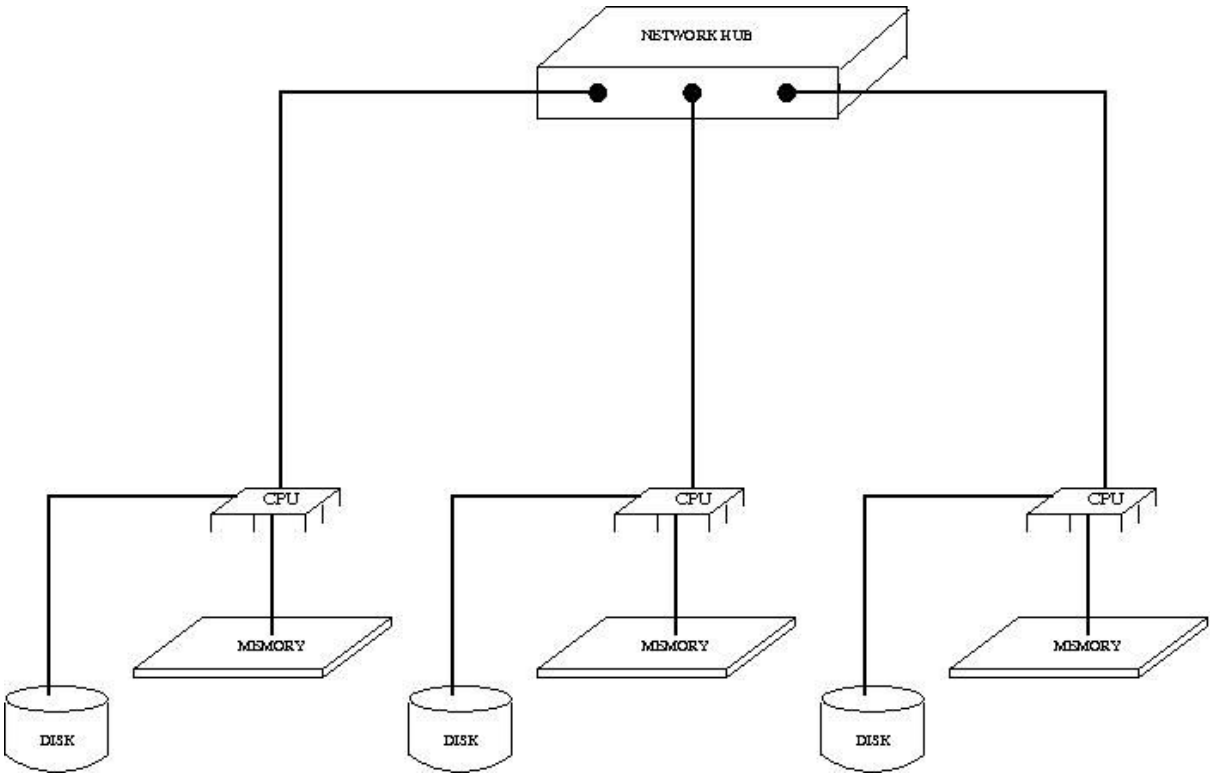
Usando el API, podemos pasar mensajes a través de las diversas computadoras que hay en la red; esta comunicación además puede ser enviada a todo el grupo de computadoras o a una pequeña parte.

One-sided communication

Cuando se transmite entre procesos, un proceso transmite un mensaje que modifica la memoria local con un valor del proceso que envía; este mecanismo mejora enormemente el tiempo de comunicación al no tener que mandar un mensaje de recepción.

Resumen y datos adicionales

Distributed-memory, al utilizar más computadores, puede realizar tareas más complejas que Shared-memory, el API utilizado puede transmitir mensajes a uno o varios computadores de la red a través de Message-passing; además se puede cambiar la memoria local de un proceso con One-sided communication, esto mejora enormemente el tiempo de comunicación.

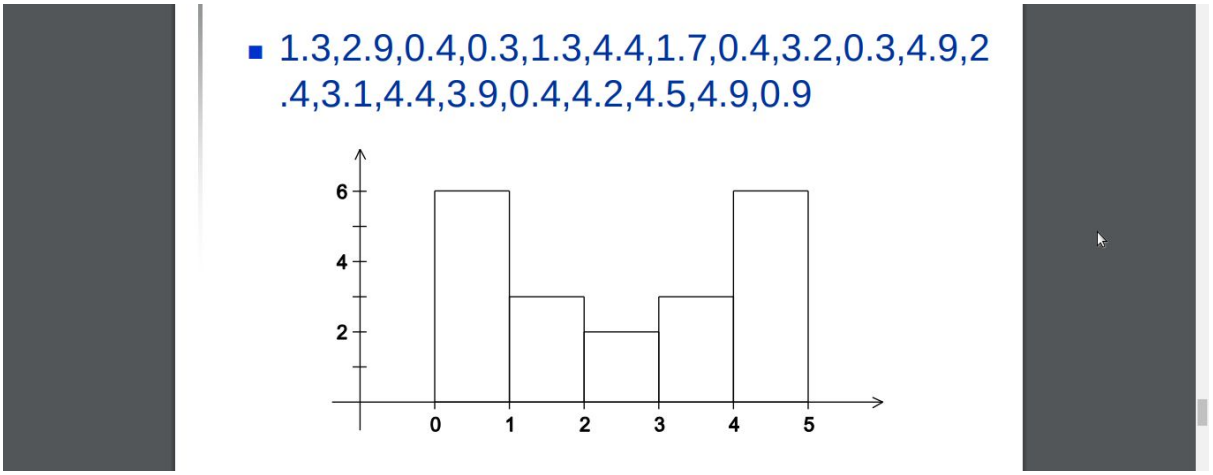


Leyes de Amdahl y Gustafson

Ley de Amdahl	Ley de Gustafson
Ley donde se afirma que no importando el número de núcleos, la mejora en aceleración no se verá deseablemente mejorada. $$S_{\text{latency}}(s) = \frac{1}{(1-p) + \frac{p}{s}}$$ Donde: ● S_{latency} es la aceleración que tendrá todo el proceso. ● s es el aumento de aceleración de una parte del proceso que se verá beneficiada por el sistema ● p es una proporción de ejecución que ha sido beneficiada por los recursos.	Ley donde se afirma con un enfoque más optimista que aumentando el número de núcleos, se mejora la aceleración. $$S(P) = P - \alpha \cdot (P-1)$$ Donde: ● S es la aceleración. ● P es el número de procesadores. ● α es la parte no paralelizable del sistema.

Se ve claramente que la ley de Amdahl es más precisa que la ley de Gustafson, ya que esta prevé el peor caso posible, mientras que la de Gustafson opta por el mejor caso, el cual siempre suele estar expuesto a varias y no solo una parte paralelizable.

Explicación del histograma



Para poder contar las apariciones en un rango de números se puede ver dos enfoques al momento de paralelizar con respecto a la memoria.

Memoria completamente compartida, donde tenemos un arreglo con las apariciones para cada rango de elementos, y…

Memoria por thread,en la cual cada thread se divide en un conjunto de threads hijos cada uno con una posición en un arreglo del respectivo thread padre donde almacenará su resultado; cada thread hijo se subdivide en threads hijos en caso que la data a contar sea muy grande.

Viéndolo en el primer enfoque el proceso se serializa al tener que hacer un lock cada vez que se quiera incrementar la data del acumulativo por posición; el segundo enfoque en cambio utiliza más memoria pero con una aceleracion mas notoria al dividir el conjunto en pequeños conjuntos que son más fáciles de contar.

En conclusión, se debe buscar una manera de dividir el problema para su paralelización así como lo señalado en la **metodología de Foster**, sea primero particionando, la comunicación que en este caso seria el arreglo del nodo padre, la aglomeración que sería la suma y el mapeamiento.

Referencias

“Shared memory,” Wikipedia, 25-Aug-2018. [Online]. Available: https://en.wikipedia.org/wiki/Shared_memory. [Accessed: 27-Aug-2018].
“Distributed memory,” Wikipedia, 22-Aug-2018. [Online]. Available: https://en.wikipedia.org/wiki/Distributed_memory. [Accessed: 27-Aug-2018].
“Ley de Amdahl,” Wikipedia, 29-Jun-2018. [Online]. Available: https://es.wikipedia.org/wiki/Ley_de_Amdahl. [Accessed: 27-Aug-2018].
“Ley de Gustafson,” Wikipedia, 24-Apr-2018. [Online]. Available: https://es.wikipedia.org/wiki/Ley_de_Gustafson. [Accessed: 27-Aug-2018].
P. E. T. E. R. PACHECO, INTRODUCTION TO PARALLEL PROGRAMMING. S.l.: MORGAN KAUFMANN PUBLISHER, 2017.