Para poder evaluar el desempeño de un sistema de computación y así poder compararlo respecto a otro necesitamos definir y medir su rendimiento. Pero, ¿Qué queremos decir con rendimiento?, ¿En base a qué parámetros podemos expresar o medir el rendimiento? ¿Cómo podemos establecer un mecanismo que nos permita comparar dos sistemas de computación?

Para poder cuantificar el rendimiento, necesitamos determinar los factores que influyen en el desempeño del equipo de cómputo y así definir una expresión que caracterice este rendimiento. Se denomina medida al valor obtenido mediante un instrumento de medición confiable. Una *medida* "proporciona una indicación cuantitativa de extensión, cantidad, dimensiones, capacidad y tamaño de algunos atributos de un proceso o producto"

El desempeño de un computador puede tener diferentes medidas de elección para diferentes usuarios. Para un usuario individual que está ejecutando un único programa, el computador con mayor rendimiento es aquel que complete la ejecución de su programa en menor tiempo. Sin embargo, para el administrador de un centro de cómputos, que tiene múltiples tareas que realizar a la vez, la de mayor rendimiento es la que le realice más tareas en menor tiempo. Como elemento común, sin embargo, se evidencia que la medida del rendimiento del computador es el tiempo. El computador que ejecute los programas en menor tiempo es la que tiene mejor rendimiento.

Un algoritmo secuencial es evaluado por su tiempo de ejecución como función del tamaño del problema. El comportamiento asintótico del tiempo de ejecución es idéntico en cualquier plataforma secuencial. En cambio, el tiempo de ejecución de un programa paralelo depende del tamaño del problema, del número de procesadores y de ciertos parámetros de comunicación de la plataforma. Es por ello que los algoritmos paralelos deben ser evaluados y analizados teniendo en cuenta también la plataforma

Además, en el mundo paralelo, adicional al tiempo de ejecución como medida de rendimiento también encontramos otras métricas como la aceleración (speedup), eficiencia (efficiency) , etc.

1. Definiciones

1.1 Rendimiento

No es sencillo definir qué es rendimiento. Por ejemplo la tarea de los ingenieros de software es diseñar e implementar programas que satisfagan los requerimientos del usuario relacionados con correctitud y rendimiento. Si ejecutamos un programa en dos computadores distintos, ustedes dirían que el computador más rápido es aquel que termina primero la ejecución del programa. Si tienen un centro de computación que dispone de dos plataformas computacionales distintas que ejecutan trabajos de muchos

usuarios, dirían que el sistema más rápido es aquel que ejecuta y completa más trabajos por día. Como usuario, a usted le interesa reducir el tiempo de respuesta y el tiempo de ejecución de su programa. Por lo tanto su percepción de rendimiento puede cambiar dependiendo de la situación.

Vamos a usar el término de ***Rendimiento*** como una medida de que tan *"bien"* un sistema, o los componentes que lo constituyen, lleva a cabo las tareas asignadas.

1.2 Escalabilidad

Otro término importante es el de ***Escalabilidad.*** Este término se refiere al estudio del cambio en las medidas de rendimiento de un sistema cuando una o varias características del sistema son variadas. Una de las características que generalmente se cambia es el número de procesadores. Otra característica relacionada con paralelismo que puede ser modificada es la ***granularidad*** del problema.

1.3 Granularidad

Granularidad consiste en la cantidad de cómputo con relación a la comunicación. En ***Granularidad Fina*** o ***Fine-grained***, las tareas individuales son relativamente pequeñas en término de tiempo de ejecución. La comunicación entre los procesadores es frecuente. En cambio en Granularidad gruesa o Coarse-grained, la comunicación entre los procesadores es poco frecuente y se realiza después de largos periodos de ejecución.

2. Métricas de Rendimiento

2.1 Aceleración (Speedup)

La medición de rendimiento en ambientes paralelos es más complejo por nuestro deseo de conocer cuánto más rápido ejecuta una aplicación en un computador paralelo. Es decir, nos interesa conocer cuál es el beneficio obtenido cuando usamos paralelismo y cuál es la aceleración que resulta por el uso de dicho paralelismo.

La aceleración puede ser definida como el radio

$$S = \frac{Tiempo\ \ Ejecución\ \ Secuencial}{Tiempo\ \ Ejecución\ \ Paralelo}$$

Existen diversas formas de expresar la aceleración que dependen de la forma en que se defina lo que es el Tiempo de Ejecución Secuencial y Paralelo.

1. Aceleración Relativa. El Tiempo de Ejecución Secuencial usado es el tiempo total de ejecución del programa paralelo cuando éste es ejecutado sobre un único procesador del computador paralelo. Por lo tanto la Aceleración Relativa de un programa paralelo Q cuando se resuelve una instancia I de tamaño n usando P procesadores es:

$$Aceleración\ \text{Re}lativa(I,P) = \frac{T(I,1)}{T(I,P)}$$

2. Aceleración Real. El Tiempo de Ejecución Secuencial usado es el tiempo del mejor programa secuencial para resolver el problema.

$$Aceleración\ Real(I,P) = \frac{T^*}{T(I,P)}$$

Formalmente la aceleración es la relación entre el tiempo de ejecución sobre un procesador secuencial y el tiempo de ejecución en múltiples procesadores. ¿Qué significa una Aceleración igual a 2?

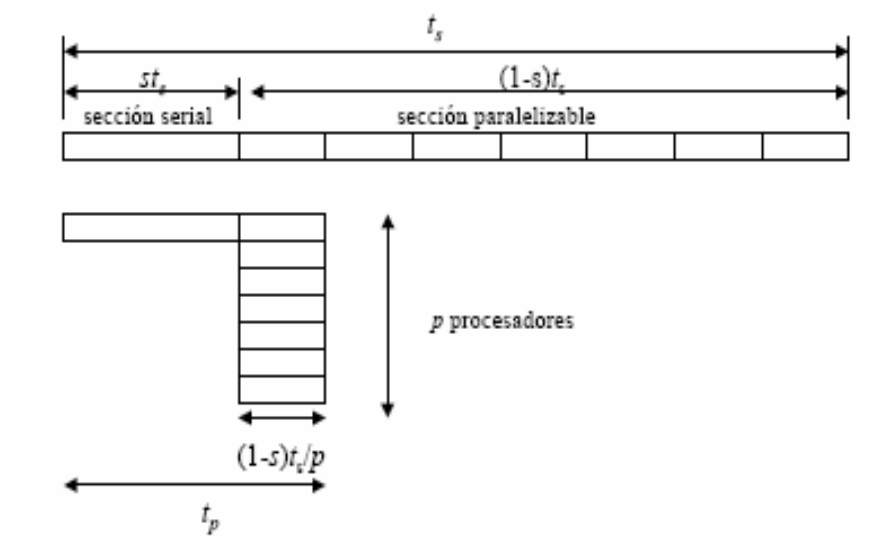
2.2 La ley de Amdahl

Un aspecto importante a considerar es que todo programa paralelo tiene una parte secuencial que eventualmente limita la aceleración que se puede alcanzar en una plataforma paralela. Por ejemplo, si el componente secuencial de un algoritmo es 1/*s* de su tiempo de ejecución, entonces la aceleración máxima posible que puede alcanzar en el computador paralelo es *s*.

El tiempo para realizar el cómputo con *P* procesadores es

$$T_P = st_s + \frac{(1-s)t_s}{P}$$

donde t_s es el tiempo de ejecución secuencial



La aceleración puede ser redefinida de la siguiente forma

$$Aceleración = \frac{1}{s + \frac{(1-s)}{P}} = \frac{P}{Ps + 1 - s} = \frac{P}{1 + (P-1)s}$$

$$\frac{P}{1 + (P-1)s} \to \frac{1}{s} \quad cuando\ P \to \infty$$

Según la ley de Amdahl, si un programa tiene un 5% de componente secuencial entonces la aceleración máxima que se puede alcanzar es de 20.

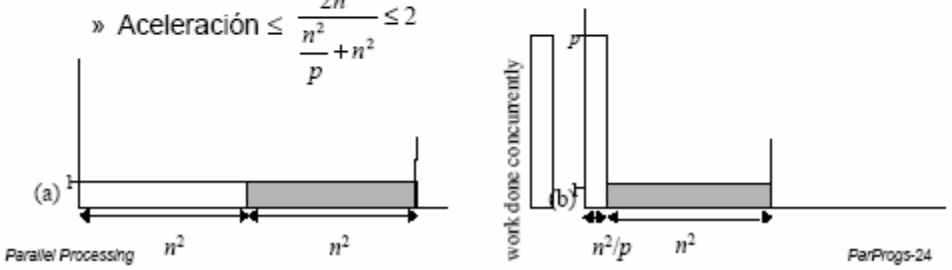
La ley de Amdahl tiene varias implicaciones:

- Para una carga dada, la aceleración máxima tiene una cota superior de 1/*s*. Al aumentar *s*, la aceleración decrecerá proporcionalmente.

- Para alcanzar buenas aceleraciones, es importante reducir *s*

Ejemplo

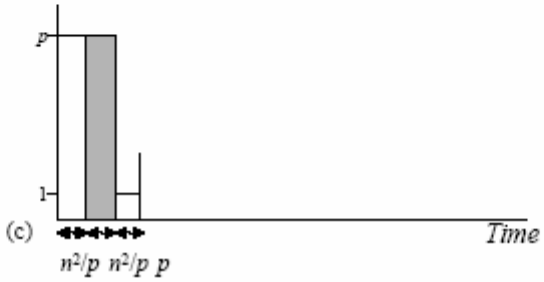
- Cálculo de dos fases
  - 1a. Fase: una malla de $n \times n$ se barre y se hacen cálculos independientes
  - 2a. Fase: la malla se barre nuevamente y se calcula la suma global
  - Tiempos de ejecución
    - » 1a. Fase: n^2/p
    - » 2a. Fase: n^2
    - » Aceleración $\leq \dfrac{2n^2}{\frac{n^2}{p}+n^2} \leq 2$



Parallel Processing ParProgs-24

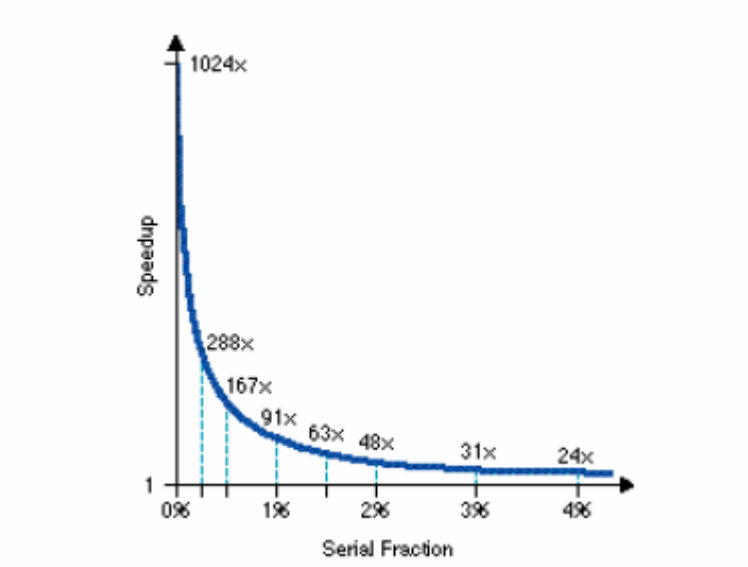
Mejora:

- Modificación
  - Divida la segunda fase en dos actividades
  - acumule las sumas privadas durante la primera fase
  - sume el total de las sumas privadas en la suma global
  - Tiempo de ejecución es $n^2/p + n^2/p + p$
- Aceleración $\dfrac{2n^2}{2n^2/p+p} = \dfrac{2n^2 p}{2n^2+p^2}$



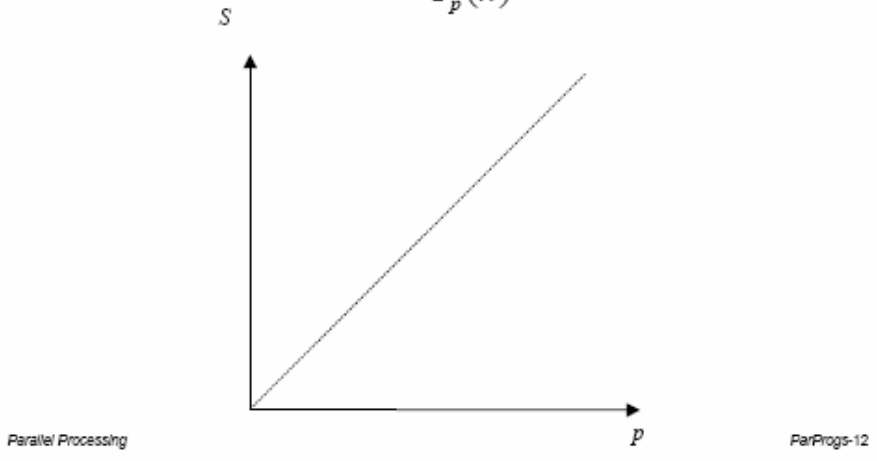
Parallel Processing ParProgs-25

En la siguiente figura podemos observar el comportamiento de la Aceleración en función de la fracción de código que no puede ser paralelizado. Vemos que aún usando una gran cantidad de procesadores (1024), con un porcentaje bajo de código serial se obtiene aceleraciones muy por debajo del número de procesadores usado. Por ejemplo, si la aplicación paralela tiene sólo un 1% de código serial, la aceleración máxima que puede lograrse es de 91, es decir que con 1024 procesadores dicha aplicación puede ejecutar 91 veces más rápido que la aplicación serial.

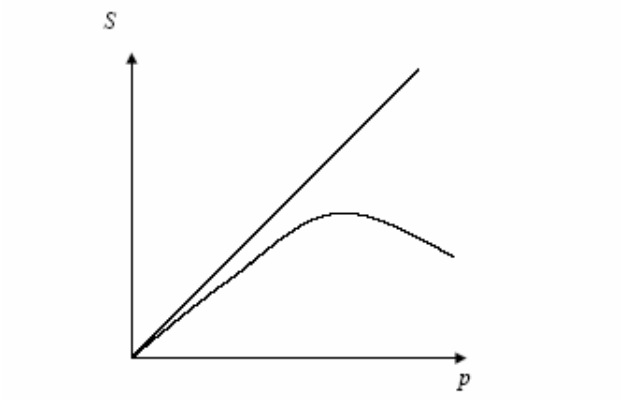


Aceleración Ideal

$$S_p^{Ideal}(n) = \frac{T^*(n)}{T_p(n)} = p$$



Lo que realmente suele suceder con la Aceleración



En los primeros tiempos de la computación paralela, se creyó que el efecto capturado por la ley de Amdahl limitaría la utilidad de la computación paralela a un grupo pequeño de aplicaciones. Sin embargo, ciertas experiencias prácticas han demostrado que esta forma de ver los programas con una parte estrictamente secuencial tiene poca relevancia en los problemas reales

Para entender esto veamos que sucede en un problema no relacionado con la computación. Supongamos que 999 obreros de 1000 que están trabajando en la construcción de una autopista están ociosos cuando uno de los obreros completa una tarea secuencial. Este hecho lo podemos ver como una falla en el manejo del proyecto y no como un atributo del problema en si. Si colocar el cemento en un punto de la autopista es un cuello de botella, se puede argumentar que la construcción de la autopista se puede hacer en varios puntos distintos al mismo tiempo. Haciendo esto se puede introducir alguna ineficiencia pues a lo mejor los camiones tienen que viajar distancias más largas para llegar al punto de trabajo pero esto nos llevará a realizar el proyecto en menos tiempo. Algo similar se puede aplicar en el caso de las soluciones paralelas de un problema en específico.

La ley de Amdahl puede ser relevante cuando un programa secuencial es paralelizado de forma incremental. De esta forma se perfila el programa secuencial para identificar los componentes con mayor demanda computacional. Estos componentes son adaptados para una ejecución paralela uno por vez hasta que se logre un rendimiento aceptable.

2.3 Ley de Gustafson-Barsis

A finales de la década de los 80, John Gustafson y su equipo de investigación en Sandia National Laboratorios se encontraban realizando investigación relacionada con el uso de Procesamiento Masivamente Paralelo y notaron que existía un alto nivel de escepticismo por el uso de estos sistemas paralelos debido a la ley de Amdahl. Según esta ley, aún en

problemas con una fracción secuencial de trabajo muy pequeña, la aceleración máxima alcanzable para un número infinito de procesadores era de solamente 1/s. Con los resultados de las investigaciones que realizaron en un sistema con 1024 procesadores demostraron que las suposiciones de la ley de Amdahl no son apropiadas para el caso de paralelismo masivo.
El problema está en que generalmente no se toma un problema de tamaño fijo y se ejecuta variando el número de procesadores. Esto se hace sólo para fines académicos y de investigación. Generalmente, el tamaño del problema crece al aumentar el número de procesadores. Cuando se dispone de un procesador más poderoso, el problema se expande para hacer uso de ese poder computacional. Los usuarios tienen control sobre aspectos como número de etapas o pasos, complejidad del operador y otros parámetros que generalmente son ajustados para permitir que el programa ejecute en un tiempo razonable de tiempo. Por lo tanto es más realista considerar que el tiempo de ejecución es constante y no que el tamaño del problema varía.

2.4 Implementación de la ley de Gustafson

Sea *n* una medida del tamaño del problema.

El tiempo de ejecución de un programa en un computador paralelo es descompuesto en:

$$a(n) + b(n) = 1$$

donde *a* es la fracción secuencial y *b* es la fracción paralela.

En un computador secuencial, el tiempo relativo será igual a $a(n) + pb(n)$ donde *p* es el número de procesadores para el caso paralelo

La aceleración es entonces

$$(a(n) + pb(n))$$

Si la función secuencial $a(n)$ disminuye a medida que se incrementa el tamaño del problema *n,* entonces la aceleración alcanzará *p* cuando se aproxima a infinito

Por lo tanto la ley de Gustafson rescata el procesamiento paralelo que no era favorecido por la ley de Amdahl.

2.5 Eficiencia

Podemos definir la eficiencia como el porcentaje de tiempo empleado en proceso efectivo

$$E = \frac{S}{P}$$

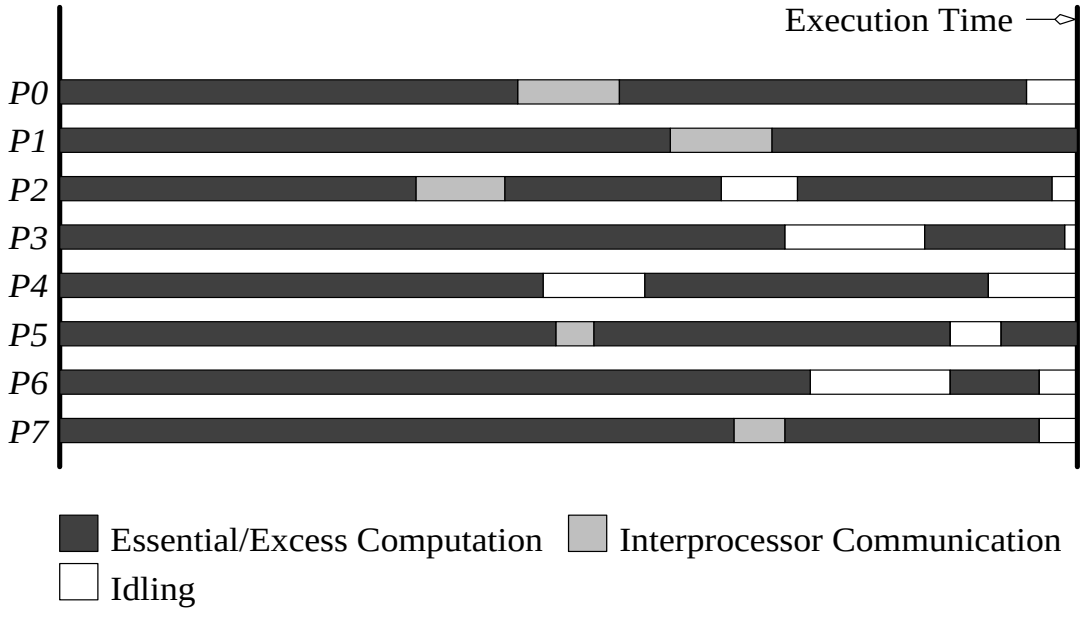
2.6 Escalabilidad

Un sistema es escalable si mantiene constante la eficiencia al aumentar el número de procesadores aumentando también el tamaño del problema

3. Fuente de *Overhead* en los programas paralelos

Si usamos dos procesadores, por qué nuestro programa no ejecuta dos veces más rápido? Porque existe una serie de tiempos que degradan el rendimiento del programa. Entre estos tiempos encontramos tiempo de comunicación, tiempo ocioso, etc.

En la figura que se encuentra a continuación se muestra el perfil de ejecución de un programa paralelo hipotético sobre 8 procesadores. El perfilador muestra tiempo ocupado en la ejecución propiamente dicha, en comunicación y ocioso.



- Comunicación entre procesos (T_{comm}): generalmente los procesadores que trabajan sobre un problema paralelo requerirán comunicarse entre si. Incluye el tiempo empleado en el envío y recepción de mensajes. El modelo de costo de comunicación que suele usarse es

$$T_{mensaje}(L) = t_s + t_w L$$

  donde t_s es el tiempo de inicialización del mensaje y t_w es el tiempo de transferencia de una palabra

- Ocioso (T_{ocio}): los procesos pueden estar ociosos en algunos instantes de tiempo debido a desbalance en la carga de trabajo asignado a cada proceso, o debido a sincronizaciones o la existencia de trozos de código que debe ser ejecutado de forma secuencial por un único procesador. Este tiempo es difícil de determinar pues depende del orden en la que se realizan las operaciones. Lo ideal es reducirlo lo más que se pueda.

- Cómputo (T_{comp}): Este tiempo corresponde a cómputo que debe ser realizado en la versión paralela y que no se ejecuta en una versión secuencial. Depende del tamaño del problema y del número de procesadores.

3.1 Cómo medimos el tiempo de ejecución de un programa paralelo?

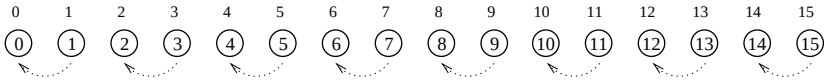
En el caso de un programa secuencial, el tiempo de ejecución es el tiempo que transcurre desde que se inicia la ejecución hasta que finaliza. En el caso de un programa paralelo es el tiempo transcurrido desde que comienza su ejecución hasta el momento en que el último procesador finaliza su ejecución (T_P).

Ejemplo

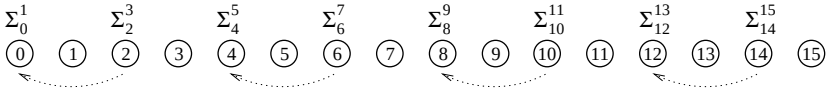
Consideremos el problema de sumar *n* números usando *n* elementos de procesamiento. Si *n* es potencia de dos, podemos realizar esta operación en log *n* pasos propagando sumas parciales a lo largo de un árbol lógico de procesadores. En la figura se muestran los pasos necesarios. Σ_i^j denota la suma de los números etiquetados desde *i* hasta *j*. Si una adición toma un tiempo constante de t_c y comunicar una palabra cuesta $t_s + t_w$, tenemos entonces que el tiempo paralelo

$$T_P = \Theta (\log n)$$

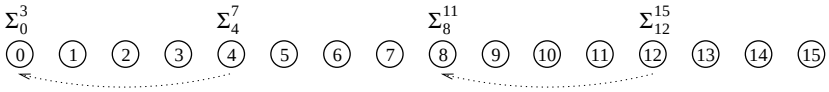
Sabemos que el tiempo secuencial es $T_S = \Theta(n)$ y la aceleración es $S = \Theta(n / \log n)$



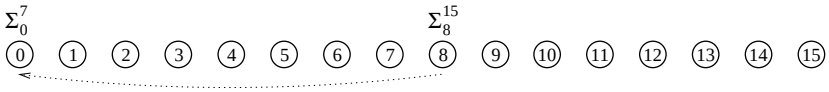
(a) Initial data distribution and the first communication step



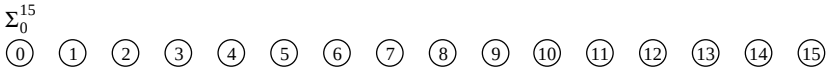
(b) Second communication step



(c) Third communication step



(d) Fourth communication step

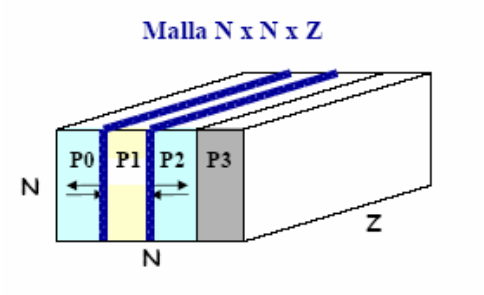


(e) Accumulation of the sum at processing element 0 after the final communication

Ejemplo

Eficiencia del algoritmo de Diferencias Finitas

Dado el problema de diferencias finitas en una mall de N x N x Z. Descomponemos la malla en la dimensión horizontal como se muestra en la siguiente figura. Cada tarea trabaja sobre una submalla con N x N/P x Z puntos.



Cada tarea lleva a cabo el mismo cómputo sobre cada punto asignado en cada paso de iteración del algoritmo. El tiempo de cómputo para cada paso es

$$T_{comp} = t_c N^2 ZP$$

donde t_c es el tiempo medio de cómputo por punto.

Cada tarea intercambia 2NZ puntos con dos vecinos, por lo tanto cada tarea envía dos mensaje con 2NZ puntos por mensaje. El tiempo de comunicación podemos expresarlo como

$$T_{comm} = 2(t_s + t_w 2NZ)$$

El tiempo total de la aplicación paralela viene dado por

$$T_P = T_{comp} + T_{comm} = t_c N^2 Z / P + 2t_s + 4t_w NZ$$

La aceleración o Speedup viene dado por

$$S = \frac{T_1}{T_P} = \frac{t_c N^2 Z}{t_c N^2 Z / P + 2t_s + 4t_w NZ}$$

y la eficiencia

$$E = \frac{S}{P} = \frac{t_c N^2 Z}{t_c N^2 Z + 2Pt_s + 4t_w PNZ}$$

Los modelos de rendimiento desarrollados hasta el momento son herramientas que pueden ser usadas para explorar y refinar el diseño de un algoritmo paralelo. Con ellos se puede realizar un análisis cualitativo de rendimiento. Por ejemplo a partir de las ecuaciones de T_P y E podemos emitir las siguientes observaciones del algoritmo de diferencias finitas:

- La eficiencia baja al incrementar el número de procesadores (P) y el costo de comunicación (t_s y t_w).

- La eficiencia sube al incrementar el tamaño del problema (N y Z) y el tiempo de cómputo de cada punto (t_c).

- T_P baja al incrementar P pero está acotado inferiormente por el costo de intercambiar trozos de la matriz.

Estas observaciones proveen visiones interesantes de las características del algoritmo. También es necesario obtener resultados cuantitativos que requieren que sustituyamos ciertos parámetros en el modelo de rendimiento por valores específicos de la arquitectura en la cuál ejecutará. Estos valores se obtienen por medio de estudios empíricos. Una vez incluidos estos valores al modelo, éste puede ser usado para contestar interrogantes como:

- ¿Cumple el algoritmo con restricciones de diseño (tiempo de ejecución, requerimientos de memoria) en la arquitectura paralela a usar?

- ¿Cómo se adapta el algoritmo? ¿Se adapta el mismo a aumentos en el tamaño del problema y en el número de procesadores?

- ¿Cómo se comporte este algoritmo con respecto a otros algoritmos para el mismo problema?

Ejemplo

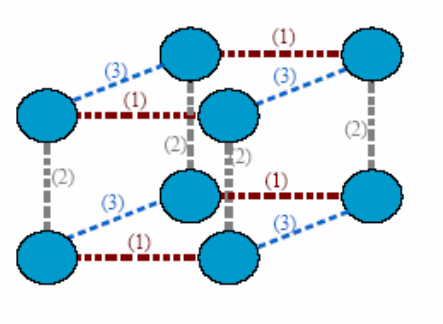
Suma de N números en un hipercubo de P procesos

A cada proceso le asignamos N/P números. Cada proceso calcula la suma parcial de sus N/P números. Por lo tanto efectúan N/P – 1 operaciones de suma. Cada proceso tiene log(P) vecinos con quines se comunica e intercambia los resultados locales. Después de intercambiar resultados, actualiza la suma.

$$T_{comp} = t_c(N/P - 1) + t_c \log P$$

$$T_{comm} = \log P(t_s + t_w)$$

$$T_P = T_{comp} + T_{comm} = t_c(N/P - 1) + (t_c + t_s + t_w)\log P$$



3. 2 Análisis de Escalabilidad

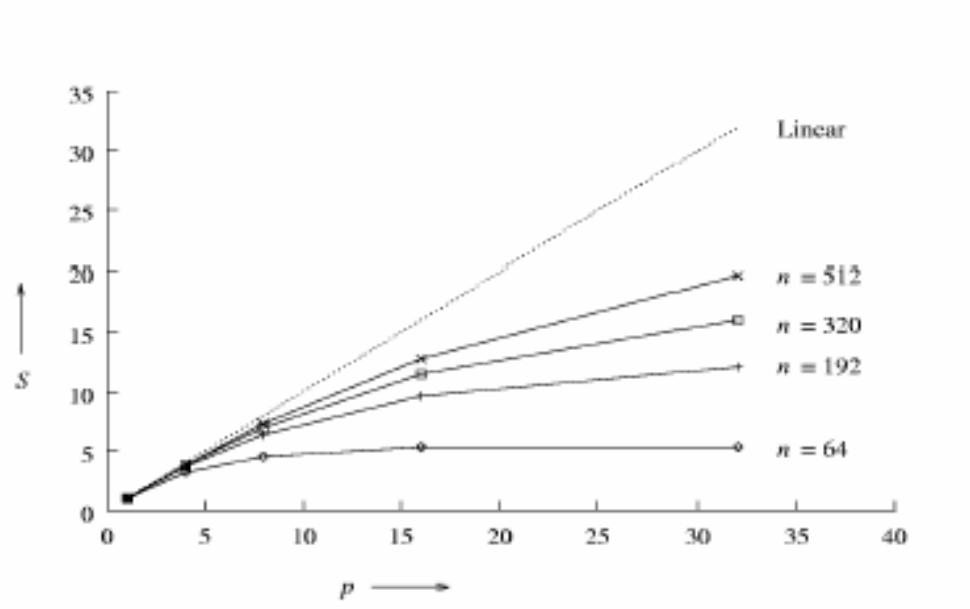
Supongamos que $t_c = (t_s + t_w) = 1$, entonces

$$T_1 = N$$

$$T_P = N/P - 1 + 2\log P \approx N/P + 2\log P$$

$$S = \frac{N}{N/P + 2\log P} \qquad E = \frac{N}{N + 2P\log P}$$

La aceleración S no se incrementa linealmente con P sino que tiende a saturarse.



Al aumentar N se produce un aumento de S y E para un P fijo. Debería ser posible incrementar tanto P como N manteniendo constante E.

En la tabla podemos observar una eficiencia de 0.80 para diferentes valores de N y P

n	*p* = 1	*p* = 4	*p* = 8	*p* = 16	*p* = 32
64	1.0	*.80*	.57	.33	.17
192	1.0	.92	*.80*	.60	.38
320	1.0	.95	.87	.71	.50
512	1.0	.97	.91	*.80*	.62

Un sistema es escalable si es posible mantener E constante incrementando simultáneamente el tamaño del problema (N) y el número de procesadores (P).

3.3 Escalabilidad con Problemas de tamaño fijo

Un aspecto importante del análisis de rendimiento es el estudio de cómo varía el rendimiento de un algoritmo cuando se varían algunos parámetros como el tamaño del problema, el número de procesadores, el costo de inicio de una comunicación, etc.

En particular nos interesa evaluar la escalabilidad de un algoritmo paralelo, es decir, que tan efectivo resultará el incremento en el número de procesadores.

Una forma para cuantificar esto es determinar como varía el tiempo de ejecución y la eficiencia cuando se incrementa el número de procesadores manteniendo fijos el tamaño del problema y el resto de los parámetros de la arquitectura. De esta forma podemos contestar interrogantes como, ¿qué tan rápido puede ser resuelto este problema en este computador? ¿Cuál es el máximo número de procesadores que puedo utilizar si deseo mantener la eficiencia en 50%?.

Es importante considerar tanto la eficiencia como el tiempo de ejecución cuando se evalúa escalabilidad

3.4 Escalabilidad con tamaño de problema escalable

Hay que tener en cuenta que los sistemas paralelos grandes no son usados solamente para resolver problemas más rápido sino que también se utilizan para la resolución de problemas más grandes. Esto nos lleva a otra alternativa en el análisis de escalabilidad que consiste en estudiar cómo la cantidad de cómputo escala con el número de procesadores manteniendo constante la eficiencia.

n	*p* = 1	*p* = 4	*p* = 8	*p* = 16	*p* = 32
64	1.0	*.80*	.57	.33	.17
192	1.0	.92	*.80*	.60	.38
320	1.0	.95	.87	.71	.50
512	1.0	.97	.91	*.80*	.62

3.5 Superlinealidad

Considere el problema de un algoritmo bubblesort paralelo. Supongamos que el tiempo secuencial es 150 segundos y el tiempo paralelo para una implementación eficiente es de 40 segundos. La aceleración es de 150/40 = 3.75. Es esto un buen logro? Qué pasa si el quicksort secuencial toma solo 30 segundos? En este caso la aceleración es de sólo 0.75.

La aceleración puede ser tan bajo como 0 (en el caso del programa paralelo que nunca termine). En teoría, la aceleración tiene una cota superior dado por el número de procesadores P. Si usamos P procesadores, nos gustaría que nuestro programa ejecute P veces más rápido. Una aceleración mayor que P es posible sólo si cada elemento de procesamiento consume menos de T_s/P resolviendo el problema. A esto se le conoce como superlineal. Una razón de esto es que la versión paralela realiza menos trabajo que el correspondiente algoritmo secuencial. Puede deberse también a factores relacionados con los recursos que utiliza.