
PROGRAMACION
PARALELA
Técnicas y Aplicaciones utilizando
Workstations en red y Computadoras
Paralelas
2da Edición
Barry Wilkinson / Michael Allen
(Capítulo 1. Secciones 1.1 y 1.2)
Traducción:
Mariela Galdamez
Revisión y compaginación:
Paola Caymes Scutari / Germán Bianchini
LICPaD (UTN-FRM) 2022

Capítulo 1 Computadoras Paralelas

Primera traducción del original *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers. Second Editions.*

Capítulo 1: Computadoras Paralelas

1.1 La demanda de velocidad computacional

Existe una demanda continua de mayor poder computacional a lo actualmente posible en sistemas informáticos. Los ámbitos que requieren gran velocidad computacional incluyen simulaciones numéricas de problemas en la ciencia y en la ingeniería. A menudo, tales problemas necesitan enormes cantidades de cálculos repetitivos en grandes volúmenes de datos para dar resultados válidos, los cuales deben ser completados dentro de un período de tiempo "razonable". Por ejemplo, en la fabricación, los cálculos y simulaciones ingenieriles deben ser conseguidos en segundos o minutos en lo posible. Una simulación que demora dos semanas para lograr un resultado es usualmente inaceptable en un ambiente de diseño, ya que el tiempo debe ser lo suficientemente corto para así el diseñador trabajar eficientemente. A medida que los sistemas se vuelven más complejos, toma cada vez más tiempo simularlos. Existen algunos problemas que tienen un plazo específico para los cálculos, como el pronóstico del tiempo. Demorar dos días para pronosticar con precisión el tiempo local para el día siguiente sería un pronóstico en vano. Algunos ámbitos, como el modelado de grandes estructuras de ADN y el pronóstico del tiempo global, son problemas que constituyen un gran desafío. Un gran desafío es aquel que no puede ser resuelto en un periodo de tiempo razonable con las computadoras actuales.

Un gran desafío es aquel que no puede ser resuelto en un periodo de tiempo razonable con las computadoras actuales. Necesitan enormes cantidades de cálculos repetitivos en grandes volúmenes de datos para dar resultados válidos, los cuales deben ser completados dentro de un período de tiempo "razonable".

El pronóstico del tiempo por computadora (predicción numérica del tiempo) es un ejemplo ampliamente citado que requiere computadoras muy potentes. La atmósfera se modela dividiéndola en regiones o celdas tridimensionales. Se utilizan ecuaciones matemáticas bastante complejas para capturar los diversos efectos atmosféricos. En esencia, las condiciones en cada celda (temperatura, presión, humedad, velocidad y dirección del viento, etc.) se calculan en intervalos de tiempo, utilizando las condiciones existentes en la misma celda y en celdas cercanas del intervalo de tiempo anterior. Los cálculos de cada celda se repiten muchas veces para modelar el paso del tiempo. La característica clave que hace que la simulación sea significativa es la cantidad de celdas que son necesarias. Para pronosticar durante días, la atmósfera se ve afectada por eventos muy distantes y, por lo tanto, es necesaria una región grande. Supongamos que consideramos toda la atmósfera global dividida en celdas de 1,6 km x 1,6 km x 1,6 km a una altura de 16 km (10 celdas de altura). Un cálculo aproximado conduce a alrededor de 5 x 10^8 celdas. Supongamos además que cada cálculo requiere 200 operaciones de punto flotante (el tipo de operación necesaria si los números tienen una parte fraccional o son elevados a una potencia). En un paso de tiempo, son necesarias 10^{11} operaciones de coma flotante. Si tuviéramos que pronosticar el clima durante 7 días utilizando intervalos de 1 minuto, habrá 10^4 pasos de tiempo y 10^{15} operaciones de punto flotante en total. Una computadora de 1 Gflops (10^9 operaciones de punto flotante/seg), con este cálculo, tomaría 10^6 segundos o más de 10 días para realizar el cálculo. Para realizar el cálculo en 5 minutos se requeriría una computadora funcionando a 3.4 Tflops (3.4 x 10^{12} operaciones de punto flotante/seg).

Una computadora tradicional tiene un solo procesador para realizar las acciones especificadas en un programa. Una forma de aumentar la velocidad computacional, que se ha considerado durante varios años, es mediante el uso de computadoras paralelas: múltiples procesadores dentro de una sola computadora (multiprocesador) o alternativamente, varias computadoras interconectadas de alguna manera, que operan juntas en un solo problema. En cualquier caso, el problema general se divide en partes, cada una de las cuales es ejecutada por un procesador independiente en paralelo. La escritura de programas para esta forma de cálculo se conoce como programación paralela. El enfoque debería proporcionar un aumento significativo en el rendimiento. La idea es que p procesadores/computadoras podrían proporcionar hasta p veces el cálculo de la velocidad de un solo procesador/computadora, sin importar cuál sea la velocidad actual del procesador/computadora, con la expectativa de que un problema se completará en una porción $1/p$ del tiempo. Por supuesto, esta es una situación ideal que rara vez se logra en la práctica. A menudo, los problemas no se pueden dividir perfectamente en partes independientes y es necesaria la interacción entre las partes, tanto para la transferencia de datos como para la sincronización de los cálculos. Sin embargo, se puede lograr una mejora sustancial, dependiendo del problema y la cantidad de paralelismo en el mismo. Lo que hace que la computación paralela sea siempre vigente es la continua mejora en la velocidad de ejecución de los procesadores, que tiene a su vez efecto en que las computadoras paralelas también sean cada vez más rápidas. Siempre habrá problemas de gran desafío que no se pueden resolver en una cantidad de tiempo razonable en las computadoras actuales.

Una forma de aumentar la velocidad computacional es mediante el uso de computadoras paralelas: múltiples procesadores dentro de una sola computadora (multiprocesador) o varias computadoras interconectadas de alguna manera (multicomputador). En cualquier caso, el problema general se divide en partes, cada una de las cuales es ejecutada por un procesador independiente en paralelo.

La escritura de programas para esta forma de cálculo se conoce como programación paralela.

Además de lograr aumentar la velocidad en la resolución de un problema existente, el uso de múltiples computadoras/procesadores a menudo permite que un problema mayor o una solución más precisa de un problema sean resueltos en un período de tiempo más razonable. Por ejemplo, calcular muchos fenómenos físicos implica dividir el problema en puntos de solución diferenciados. Como hemos mencionado, pronosticar el clima implica dividir el aire en una cuadrícula tridimensional de puntos de solución. Utilizar varias computadoras o un multiprocesador a menudo permitirá calcular más puntos de solución en un tiempo determinado y, por tanto, una solución más precisa. Un factor relacionado es que, muy a menudo, varias computadoras tienen más memoria principal total que una sola computadora, lo que permite abordar problemas que requieren grandes cantidades de memoria principal.

Incluso si un problema se puede resolver en un tiempo razonable, surgen situaciones en las que el mismo problema debe evaluarse varias veces con diferentes valores de entrada. Esta situación es especialmente aplicable a computadoras paralelas ya que, sin ninguna alteración del programa, se pueden ejecutar múltiples instancias del mismo programa en diferentes procesadores/computadoras simultáneamente. Los problemas de simulación a menudo se

incluyen en esta categoría. El código de simulación simplemente se ejecuta en computadoras separadas simultáneamente, pero con diferentes valores de entrada.

El uso de múltiples computadoras/procesadores permite:

- aumentar la velocidad en la resolución de un problema existente.

- que un problema mayor o una solución más precisa de un problema sean resueltos en un período de tiempo más razonable.

- resolver problemas que deben evaluarse varias veces con diferentes valores de entrada, ya que se pueden ejecutar múltiples instancias del mismo programa en diferentes procesadores/computadoras simultáneamente.

Finalmente, la aparición de Internet y la red mundial ha generado una nueva área para el cómputo paralelo. Por ejemplo, los servidores web a menudo deben manejar miles de solicitudes por hora de los usuarios. Una computadora multiprocesador, o más probablemente en la actualidad varias computadoras conectadas juntas como un clúster se utilizan para atender la solicitud. Las solicitudes individuales son atendidas por diferentes procesadores o computadoras simultáneamente.

La computadora paralela no es una idea nueva, de hecho, es una idea muy antigua. A pesar de la larga historia, Flynn y Rudd (1996) escribieron que "el impulso continuo por sistemas de mayor y mayor rendimiento ... nos lleva a una simple conclusión: el futuro es paralelo".

1.2 El potencial de aumentar la velocidad computacional

De aquí en más, el número de procesos o procesadores se identificará como p. Usaremos el término “multiprocesador” para incluir todos los sistemas informáticos en paralelo que contienen más de un procesador.

1.2.1 Factor de Speedup

Quizás el primer punto de interés al desarrollar soluciones en un multiprocesador es la pregunta de cuánto más rápido el multiprocesador resuelve el problema en consideración. Al hacer esta comparación, se usaría la mejor solución en un solo procesador, es decir, el mejor algoritmo secuencial en un uniprocesador para compararlo con el algoritmo paralelo bajo investigación en el multiprocesador. El factor de aceleración (comúnmente llamado factor de Speedup) , $S(p)$[1], es una medida del rendimiento relativo, que se define como:

$$S(p) = \frac{t_S}{t_p}$$

En esta expresión t_S representa el tiempo de ejecución del mejor algoritmo secuencial ejecutándose en un solo procesador y t_p representa el tiempo de ejecución para resolver el mismo problema en un multiprocesador. $S(p)$ constituye entonces el incremento en la velocidad usando el multiprocesador. Es importante tener en cuenta que el algoritmo para la

[1] Este factor normalmente es una función de *p* y el número de datos a procesar, *S(p,n).* Introduciremos este último parámetro más adelante. Por ahora, la única variable es *p.*

implementación en paralelo puede no ser igual al algoritmo en el uniprocesador (lo cual ocurre con frecuencia, y resultan ser diferentes).

En un análisis teórico, el factor *Speedup* puede también ser medido en términos de pasos computacionales:

$$S(p) = \frac{\text{número de pasos computacionales con un uniprocesador}}{\text{número de pasos computacionales con } p \text{ procesadores}}$$

Para cálculos secuenciales, es común comparar distintos algoritmos considerando la complejidad computacional, que podría también ser aplicada a los algoritmos paralelos y al factor de Speedup. Sin embargo, considerar solamente los pasos computacionales podría no ser útil ya que las implementaciones en paralelo pueden necesitar gestionar la comunicación entre las partes, siendo esto mucho más laborioso que los pasos computacionales.

El valor máximo de *Speedup* es usualmente p con p procesadores (tratándose de un *Speedup* lineal). El aumento de *Speedup* p sería alcanzado cuando el cálculo puede ser dividido en procesos de igual duración, con un proceso mapeado (asignado o funcionando) en un procesador independiente (relación 1 a 1) y sin sobrecarga adicional en la solución paralela (real o aparente generada por las comunicaciones, la división de tareas, la asignación, etc.).

$$S(p) \leq \frac{t_S}{t_S/p} = p$$

Un ***Speedup*** **superlineal**, donde $S(p) > p$, puede encontrarse en alguna ocasión, pero usualmente esto se debe a algoritmos secuenciales por debajo del nivel óptimo, a una característica única de la arquitectura del sistema que favorece el paralelismo, o a una naturaleza indeterminada del algoritmo. Generalmente, si un algoritmo paralelo puramente determinista lograra aumentar el *Speedup* más que *p* veces sobre el algoritmo secuencial actual, el algoritmo paralelo podría emularse en un solo procesador, una parte paralela tras otra, lo que sugeriría que el algoritmo secuencial original no era óptimo.

Una causa común del *Speedup* superlineal es la memoria extra en un sistema multiprocesador. Por ejemplo, supongamos que la memoria principal asociada a cada procesador en un sistema multiprocesador es la misma que la asociada al procesador en un sistema uniprocesador. Entonces, la memoria principal total en el sistema multiprocesador es mayor y puede contener más datos del problema en cualquier momento, lo que conduce a menos tráfico en el disco.

Eficiencia. A veces es útil saber cuánto tiempo son utilizados los procesadores en el cálculo, lo cual se puede encontrar con la eficiencia, E, definida como:

$$\mathrm{E} = \frac{\text{tiempo de ejecución en uniprocesador}}{\text{tiempo de ejecución en multiprocesador x número de procesadores}} = \frac{t_S}{t_p \; x \; p}$$

$$\mathrm{E} = \frac{S(p)}{p} \times 100\%$$

Si E es igual a 50%, los procesadores están siendo utilizados la mitad del tiempo, en promedio. Un 100% de eficiencia ocurre cuando todos los procesadores son utilizados todo el tiempo y el factor *Speedup* $S(p)$ es igual a p.

1.2.2 ¿Cuál es el máximo Speedup?

Múltiples factores aparecerán como sobrecarga en la versión paralela, limitando el *Speedup* notablemente:

1. Periodos donde no todos los procesadores pueden realizar trabajo útil y son simplemente ociosos.
2. Cálculos extra en la versión paralela que no aparecen en la versión secuencial; por ejemplo, recalcular constantes localmente.
3. Tiempo de comunicación entre procesos.

Es razonable esperar que alguna parte de un cálculo no pueda ser dividido en procesos concurrentes y deba ser ejecutado secuencialmente. Por ejemplo, un periodo de inicialización o el periodo anterior a que se configuren los procesos concurrentes, sólo un procesador está trabajando de manera útil.

Suponiendo que habrá partes del cálculo que no pueden ser divididas en procesos concurrentes, la situación ideal sería que todos los procesadores disponibles trabajaran simultáneamente el resto del tiempo. Si la fracción del cálculo donde no se puede dividir en tareas concurrentes es f, y no hay sobrecarga ocasionada por la división del cálculo en partes concurrentes, el tiempo para realizar el cálculo con p procesadores está dado por $ft_s + (1-f)t_s/p$, ilustrado en la Figura 1.1. El factor *Speedup* puede redefinirse entonces de la siguiente manera:

$$S(p) = \frac{t_s}{ft_s + (1-f)t_s/p} = \frac{p}{1+(p-1)f}$$

La ecuación es conocida como ley de Amdahl[2]. La Figura 1.2 muestra $S(p)$ relacionado al número de procesadores y a f. En efecto, se indica una mejora en la velocidad. Sin embargo, la fracción del cálculo ejecutada por procesos concurrentes debe ser una fracción de tamaño importante en el todo si se desea alcanzar un aumento de *Speedup* considerable. Aún con un número infinito de procesadores, el *Speedup* máximo está limitado por $1/f$.

$$\lim_{p\to\infty} S(p) = \frac{1}{f}$$

[2](Amdahl, 1967.

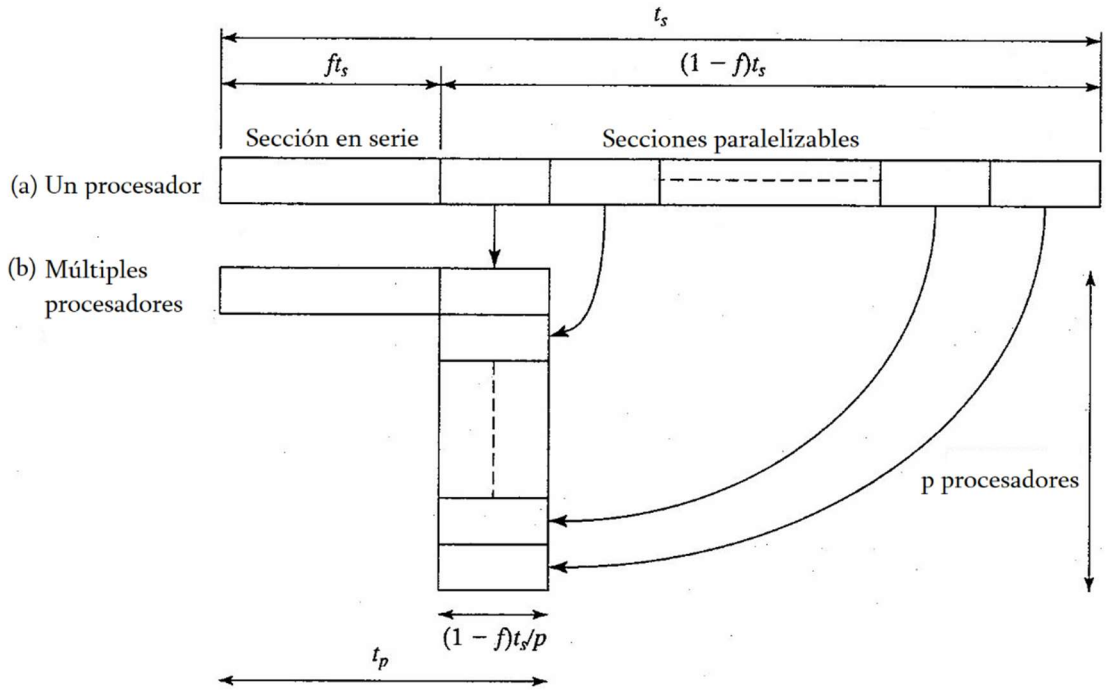

Figura 1.1 Paralelizar problemas secuenciales - Ley de Amdahl

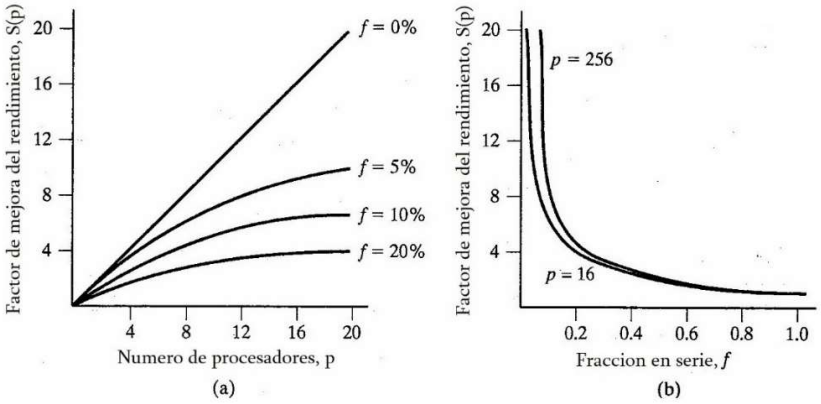

Figura 1.2 (a) S(p) relacionado al numero de procesadores. (b) S(p) relacionado a la fraccion en serie, *f*.

En ciertas circunstancias es posible alcanzar mejoras considerables. Por ejemplo, un *Speedup* superlineal puede ocurrir en algoritmos de búsqueda. En problemas de búsqueda ejecutados para buscar exhaustivamente la solución, supongamos que el espacio de solución se divide entre los procesadores para que cada uno realice una búsqueda independiente. En una implementación secuencial, los distintos espacios de búsqueda se pueden realizar uno tras otro. En una implementación paralela pueden hacerse simultáneamente, y un procesador puede encontrar la solución casi inmediatamente. En la versión secuencial, supongamos que *x* subespacios son buscados y luego, la solución es encontrada en el instante *Δt* en el siguiente subespacio buscado. El número de subespacios de búsqueda previos, *x*, no está determinado y dependerá del problema. En la versión paralela, la solución es encontrada inmediatamente en el instante *Δt*, ilustrado en la Figura 1.3.

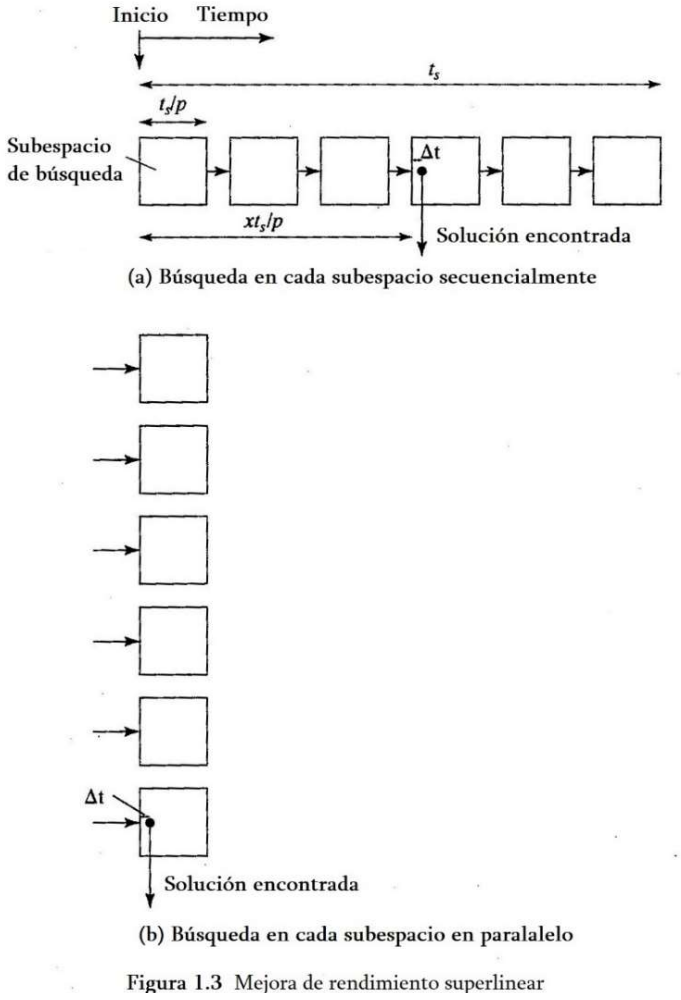

Figura 1.3 Mejora de rendimiento superlinear

En este caso, el factor Speedup está dado por:

$$S(p) = \frac{\left(x \times \frac{t_S}{p}\right) + \Delta t}{\Delta t}$$

El peor caso de la búsqueda secuencial ocurre cuando la solución se encuentra en el último subespacio de búsqueda, y la versión paralela ofrece el mejor beneficio:

$$S(p) = \frac{\left(\frac{p-1}{p}\right) \times t_S + \Delta t}{\Delta t} \rightarrow \infty \text{ con } \Delta t \text{ tendiendo a } 0$$

El caso menos ventajoso para la versión paralela sucede cuando la solución se encuentra en el primer subespacio de búsqueda de la búsqueda secuencial:

$$S(p) = \frac{\Delta t}{\Delta t} = 1$$

En definitiva, para una situación como la anterior, el *Speedup* dependerá de en qué posición relativa se encuentre la solución dentro del subespacio.

Escalabilidad. El rendimiento de un sistema dependerá del tamaño del sistema (número de procesadores) y generalmente, cuanto más grande sea el sistema mejor será el rendimiento alcanzable. No obstante, esta característica tiene un costo asociado. Escalabilidad es un término bastante impreciso. Este término es usado para indicar que un diseño de hardware permite al sistema ser incrementado en tamaño y, al hacerlo, obtener un mayor rendimiento. Esto se puede describir como escalabilidad en arquitectura o hardware. La escalabilidad también se utiliza para indicar que un algoritmo paralelo puede recibir más datos con un bajo y limitado incremento en pasos computacionales. Esto se puede describir como escalabilidad algorítmica.

Escalabilidad en hardware: indica que un diseño de hardware permite al sistema ser incrementado en tamaño, obteniendo mayor rendimiento.

Escalabilidad algorítmica: indica que un algoritmo paralelo puede recibir más datos con un bajo y limitado incremento en pasos computacionales.

Por supuesto, sería deseable que todos los sistemas multiprocesadores sean escalables en hardware, pero esto dependerá fuertemente del diseño del sistema. Normalmente, al añadir procesadores al sistema, la red de interconexión debe ser expandida. Como resultado tenemos mayor demora en comunicación y mayor contención de recursos. En consecuencia, la eficiencia del sistema, E, se reduce. El objetivo básico de la mayoría de diseños multiprocesador es lograr la escalabilidad, y esto se refleja en la gran cantidad de redes de interconexión que han sido ideadas.

La escalabilidad combinada (hardware/algorítmica) sugiere que los problemas de mayor tamaño pueden adaptarse a sistemas de mayor tamaño para una arquitectura y algoritmo particular. Aumentar el tamaño del sistema significa claramente agregar procesadores. Sin embargo, aumentar el tamaño del problema requiere ciertas aclaraciones. Intuitivamente, pensaríamos en la cantidad de elementos de datos que se procesan en el algoritmo como una medida de tamaño. Sin embargo, duplicar el tamaño del problema no necesariamente duplicaría el número de pasos computacionales. Dependerá del problema. Por ejemplo, sumar dos matrices, tiene este efecto, pero multiplicar matrices no. El número de pasos computacionales para multiplicar matrices se cuadruplica. Por lo tanto, escalar diferentes problemas implicaría diferentes requisitos computacionales. Una definición alternativa del tamaño del problema es equiparar el tamaño del problema con el número de pasos básicos en el mejor algoritmo secuencial. Por supuesto, incluso con esta definición, si aumentamos el número de datos, aumentaremos el tamaño del problema.

En los siguientes capítulos, además del número de procesadores, p, también usaremos n como el número de elementos de datos de entrada en un problema[3]. Tanto p como n, por lo general se pueden alterar en un intento de mejorar el rendimiento. Alterar p modifica el tamaño del sistema informático y alterar n modifica el tamaño del problema. Por lo general, aumentar el tamaño del problema mejora el rendimiento relativo porque se puede lograr más paralelismo.

Gustafson presentó un argumento basado en conceptos de escalabilidad para mostrar que la ley de Amdahl no era tan significativa como se suponía al principio para determinar los límites de Speedup potencial[4]. Gustafson hace la observación de que, en la práctica, un multiprocesador

[3] Para matrices, consideramos matrices de *n x n*.

[4] Gustafson, 1988.

más grande normalmente permite abordar un problema de mayor tamaño en un tiempo de ejecución razonable. Por lo tanto, en la práctica, el tamaño del problema seleccionado con frecuencia depende del número de procesadores disponibles. En lugar de asumir que el tamaño del problema es fijo, es igualmente válido asumir que el tiempo de ejecución en paralelo es fijo. A medida que aumenta el tamaño del sistema (p aumenta), el tamaño del problema aumenta para mantener un tiempo de ejecución en paralelo constante. Al aumentar el tamaño del problema, Gustafson también argumenta que la sección serial del código normalmente es fija y no aumenta (considerablemente) con el tamaño del problema.

Utilizando el tiempo de ejecución en paralelo constante como restricción, el factor de *Speedup* resultante será numéricamente diferente del factor de *Speedup* de Amdahl y se denomina factor de *Speedup* escalado (factor de *Speedup* cuando se escala el problema). Para el factor de *Speedup* escalado de Gustafson, el tiempo de ejecución en paralelo, $\mathrm{t_p}$ es constante en lugar del tiempo de ejecución en serie, t_s, en la ley de Amdahl. El factor de *Speedup* escalado se define como:

$$S_S(p) = \frac{ft_S + (1-f)t_S}{ft_S + \frac{(1-f)t_S}{p}} = \frac{p + (1-p)ft_S}{1} = p + (1-p)ft_S$$

el cual es llamado ley de Gustafson. Hay dos supuestos en esta ecuación: el tiempo de ejecución en paralelo es constante y la parte que debe ejecutarse secuencialmente, ft_S, también es constante y no es una función de p. La observación de Gustafson aquí es que el factor de *Speedup* escalado es una línea de pendiente negativa $(1-p)$ en lugar de la rápida reducción ilustrada previamente en la Figura 1.2 (b). Por ejemplo, supongamos que tenemos una sección en serie del 5% y 20 procesadores: el aumento de *Speedup* es 0.05 + 0.95 (20) = 19.05 según la fórmula, en lugar de 10.26 según la ley de Amdahl. Gustafson cita ejemplos de factores de *Speedup* de 1021, 1020 y 1016 que se han logrado en la práctica con un sistema de 1024 procesadores en problemas numéricos y de simulación.

Aparte de escalar con el tamaño del problema constante (suposición de Amdahl) y escalar con restricciones de tiempo (suposición de Gustafson), escalar podría ser escalar con restricciones de memoria. Al escalar con restricción de memoria, el problema se escala para ajustarse a la memoria disponible. A medida que aumenta el número de procesadores, normalmente la memoria crece en proporción. Esta forma puede conducir a aumentos significativos en el tiempo de ejecución[5].

1.2.3 Computación con paso de mensajes

El análisis hasta ahora no tiene en cuenta el paso de mensajes, el cual puede ser una sobrecarga muy significativa en el cálculo. En la programación de paso de mensajes, forma de programación paralela, los mensajes se envían entre procesos para pasar datos y con fines de sincronización. Así,

$$\mathrm{t_p} = \mathrm{t_{comm}} + \mathrm{t_{comp}}$$

donde $\mathrm{t_{comm}}$ es el tiempo de comunicación y $\mathrm{t_{comp}}$ es el tiempo de cómputo. A medida que dividimos el problema en partes paralelas, el tiempo de cómputo de las partes paralelas generalmente disminuye ya que las partes se vuelven más pequeñas, y el tiempo de comunicación entre las partes generalmente aumenta (ya que hay más partes que se comunican). En algún momento, el tiempo de comunicación dominará el tiempo de ejecución general y el tiempo de

[5] Singh, Hennessy y Gupta, 1993.

ejecución paralelo de hecho aumentará. Es esencial reducir la sobrecarga de comunicación debido al tiempo significativo tomado por la comunicación entre procesos. El aspecto de comunicación en la solución paralela generalmente no está presente en la solución secuencial y se considera como sobrecarga.

La relación

$$\text{cómputo/comunicación} = \frac{\text{tiempo de cómputo}}{\text{tiempo de comunicación}} = \frac{\mathrm{t_{comp}}}{\mathrm{t_{comm}}}$$

se puede utilizar como un indicador. Más adelante desarrollaremos ecuaciones para el tiempo de cálculo y el tiempo de comunicación en términos de número de procesadores (p) y número de elementos de datos (n) para algoritmos y problemas bajo consideración para manejar la posible mejora de *Speedup* y el efecto de aumentar p y n.

En una situación práctica puede que no tengamos mucho control sobre el valor de p, es decir, el tamaño del sistema que podemos utilizar (excepto que podríamos asignar más de un proceso del problema en un procesador, aunque esto no suele ser beneficioso). Supongamos que, por ejemplo, para algún valor de p, un problema requiere $c_1 n$ cómputos y $c_2 n^2$ comunicaciones. Claramente, a medida que n aumenta, el tiempo de comunicación aumenta más rápido que el tiempo de cálculo. Esto se puede ver claramente a partir de la relación cómputo/comunicación, $(c_1/c_2 n)$, que se puede convertir a notación de complejidad de tiempo a fin de eliminar las constantes. Por lo general, queremos que la relación cómputo/comunicación sea lo más alta posible, es decir, alguna función altamente creciente de n para que el aumento del tamaño del problema disminuya los efectos del tiempo de comunicación. Por supuesto, éste es un asunto complejo con muchos factores. Finalmente, sólo se puede verificar la velocidad de ejecución al ejecutar el programa en un sistema multiprocesador real, y se asume que esto se hará.