Capítulo 6

El rendimiento de los sistemas paralelos

6.1. Magnitudes y medidas del rendimiento

En esta sección se definirán algunas de las medidas más utilizadas a la hora de determinar el rendimiento de una arquitectura paralela. Así, se introducen los conceptos de: *speed-up*, eficiencia de un sistema, utilización, redundancia, etc. Esta sección y la siguiente se encuentran completas en [Hwa93].

6.1.1. Eficiencia, redundancia, utilización y calidad

Ruby Lee (1980) definió varios parámetros para evaluar el cálculo paralelo. A continuación se muestra la definición de dichos parámetros.

Eficiencia del sistema. Sea $O(n)$ el número total de operaciones elementales realizadas por un sistema con n elementos de proceso, y $T(n)$ el tiempo de ejecución en pasos unitarios de tiempo. En general, $T(n) < O(n)$ si los n procesadores realizan más de una operación por unidad de tiempo, donde $n \geq 2$. Supongamos que $T(1) = O(1)$ en un sistema mono-procesador. El *factor de mejora del rendimiento* (*speed-up*) se define como

$$S(n) = T(1)/T(n)$$

La *eficiencia del sistema* para un sistema con n procesadores se define como

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{nT(n)}$$

La eficiencia es una comparación del grado de *speed-up* conseguido frente al valor máximo. Dado que $1 \leq S(n) \leq n$, tenemos $1/n \leq E(n) \leq 1$.

La eficiencia más baja ($E(n) \to 0$ corresponde al caso en que todo el programa se ejecuta en un único procesador de forma serie. La eficiencia máxima ($E(n) = 1$) se obtiene cuando todos los procesadores están siendo completamente utilizados durante todo el periodo de ejecución.

Escalabilidad. Un sistema se dice que es *escalable* para un determinado rango de procesadores $[1..n]$, si la eficiencia $E(n)$ del sistema se mantiene constante y cercana a la unidad en todo ese rango. Normalmente todos los sistemas tienen un determinado número de procesadores a partir del cual la eficiencia empieza a disminuir de forma más o menos brusca. Un sistema es más escalable que otro si este número de procesadores, a partir del cual la eficiencia disminuye, es menor que el otro.

No hay que confundir escalabilidad con ampliabilidad. Un sistema es ampliable si físicamente se le pueden poner más módulos (más memorias, más procesadores, más tarjetas de entrada/salida, etc). Que un sistema sea ampliable no significa que sea escalable, es decir, que un sistema sea capaz de ampliarse con muchos procesadores no significa que el rendimiento vaya a aumentar de forma proporcional, por lo que la eficiencia no tiene por qué mantenerse constante y por tanto el sistema podría no ser escalable.

Redundancia y utilización. La *redundancia* en un cálculo paralelo se define como la relación entre $O(n)$ y $O(1)$:

$$R(n) = O(n)/O(1)$$

Esta proporción indica la relación entre el paralelismo software y hardware. Obviamente, $1 \leq R(n) \leq n$. La *utilización del sistema* en un cálculo paralelo se define como

$$U(n) = R(n)E(n) = \frac{O(n)}{nT(n)}$$

La utilización del sistema indica el porcentaje de recursos (procesadores, memoria, recursos, etc.) que se utilizan durante la ejecución de un programa paralelo. Es interesante observar la siguiente relación: $1/n \leq E(n) \leq U(n) \leq 1$ y $1 \leq R(n) \leq 1/E(n) \leq n$.

Calidad del paralelismo. La *calidad* de un cálculo paralelo es directamente proporcional al *speed-up* y la eficiencia, inversamente proporcional a la redundancia. Así, tenemos

$$Q(n) = \frac{S(n)E(n)}{R(n)} = \frac{T^3(1)}{nT^2(n)O(n)}$$

Dado que $E(n)$ es siempre una fracción y $R(n)$ es un número entre 1 y n, la calidad $Q(n)$ está siempre limitada por el *speed-up* $S(n)$.

Para terminar con esta discusión acerca de los índices del rendimiento, usamos el *speed-up* $S(n)$ para indicar el grado de ganancia de velocidad de una computación paralela. La eficiencia $E(n)$ mide la porción útil del trabajo total realizado por n procesadores. La redundancia $R(n)$ mide el grado del incremento de la carga.

La utilización $U(n)$ indica el grado de utilización de recursos durante un cálculo paralelo. Finalmente, la calidad $Q(n)$ combina el efecto del *speed-up*, eficiencia y redundancia en una única expresión para indicar el mérito relativo de un cálculo paralelo sobre un sistema.

6.1.2. Perfil del paralelismo en programas

El grado de paralelismo refleja cómo el paralelismo software se adapta al paralelismo hardware. En primer lugar caracterizaremos el perfil del paralelismo de un programa

para a continuación introducir los conceptos de paralelismo medio y definir el *speed-up* ideal en una máquina con recursos ilimitados.

Grado de paralelismo. *Es el número de procesos paralelos en los que se puede dividir un programa en un instante dado.* La ejecución de un programa en un ordenador paralelo puede utilizar un número diferente de procesadores en diferentes periodos de tiempo. Para cada periodo de tiempo, el número de procesadores que se puede llegar a usar para ejecutar el programa se define como el *grado de paralelismo* (GDP).

A la gráfica 6.1, que muestra el GDP en función del tiempo, se la denomina *perfil del paralelismo* de un programa dado. Por simplicidad, nos concentraremos en el análisis de los perfiles de un único programa. En la figura se muestra un ejemplo de perfil del paralelismo del algoritmo divide y vencerás.

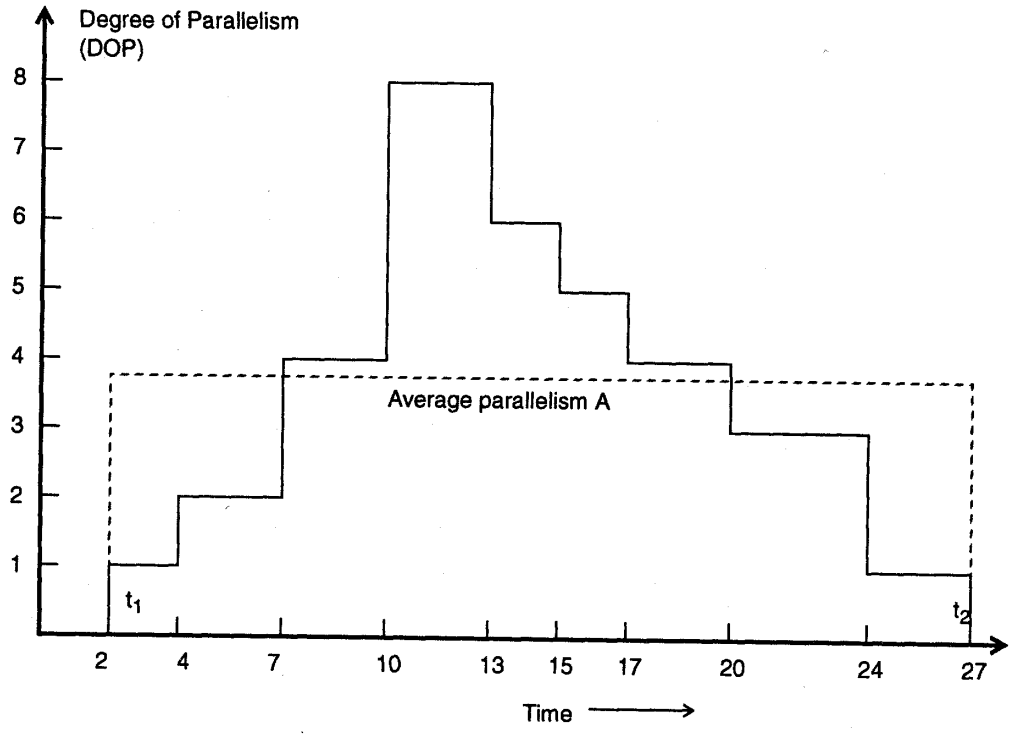


Figura 6.1: Perfil del paralelismo de un algoritmo del tipo divide y vencerás.

Las fluctuaciones en el perfil durante un periodo de observación depende de la estructura del algoritmo, la optimización del programa, la utilización de recursos, y las condiciones de ejecución del sistema donde se ejecuta el programa.

Paralelismo medio. Consideremos un procesador paralelo compuesto por n elementos de proceso homogéneos. Llamamos m al paralelismo máximo en un perfil. En el caso ideal $n \gg m$. Llamamos Δ a la *capacidad de cómputo* de un procesador, expresada en MIPS o Mflops, sin considerar las penalizaciones debidas al acceso a memoria, latencia de las comunicaciones, o sobrecarga del sistema. Cuando i procesadores están ocupados durante un periodo de tiempo, se tiene que GDP$= i$ en ese periodo.

La cantidad de trabajo realizado, a la que llamaremos W, es proporcional al área bajo la curva de perfil paralelo:

$$W = \Delta \int_{t_1}^{t_2} \mathrm{GDP}(t)\, dt$$

Esta integral se calcula frecuentemente mediante el siguiente sumatorio:

$$W = \Delta \sum_{i=1}^{m} i \cdot t_i$$

donde t_i es el tiempo que GDP$= i$ y $\sum_{i=1}^{m} t_i = t_2 - t_1$ es el tiempo total de ejecución.

El *paralelismo medio*, que llamaremos A, será por tanto

$$A = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} \mathrm{GDP}(t)\, dt$$

o en su forma discreta

$$A = \frac{\sum_{i=1}^{m} i \cdot t_i}{\sum_{i=1}^{m} t_i} \tag{6.1}$$

***Speed-up* asintótico**. Si denotamos por $W_i = i\Delta t_i$ al trabajo realizado cuando GDP$= i$, entonces podemos escribir $W = \sum_{i=1}^{m} W_i$. Esto está suponiendo que no hay sobrecarga de ningún tipo, es decir, se trata del caso ideal de paralelización.

El tiempo de ejecución de W_i sobre un único procesador es $t_i(1) = W_i/\Delta$. El tiempo de ejecución de W_i sobre k procesadores es $t_i(k) = W_i/k\Delta$. Con un número infinito de procesadores disponibles, $t_i(\infty) = W_i/i\Delta$, para $1 \leq i \leq m$. Así, podemos escribir el *tiempo de respuesta* para un procesador e infinitos procesadores como:

$$T(1) = \sum_{i=1}^{m} t_i(1) = \sum_{i=1}^{m} \frac{W_i}{\Delta}$$

$$T(\infty) = \sum_{i=1}^{m} t_i(\infty) = \sum_{i=1}^{m} \frac{W_i}{i\Delta}$$

El *speed-up asintótico* S_∞ se define como el cociente de $T(1)$ y $T(\infty)$, es decir, es un parámetro que mide la aceleración del tiempo de cálculo por el hecho de poder paralelizar al máximo la aplicación:

$$S_\infty = \frac{T(1)}{T(\infty)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i}} \tag{6.2}$$

Si comparamos esta fórmula (6.2) con la del paralelismo medio, A (6.1), se observa que $S_\infty = A$ en el caso ideal. En general, $S_\infty \leq A$ si se consideran las latencias debidas a las comunicaciones y otras sobrecargas del sistema. Observar que tanto S_∞ como A están definidos bajo la suposición de que $n = \infty$ o $n \gg m$.

Paralelismo disponible. Como ya vimos en las primeras secciones de este tema, existe un amplio grado de paralelismo potencial en los programas. Los códigos científicos presentan un alto grado de paralelismo, debido al paralelismo inherente de los propios datos. Manoj Kumar (1988) indica que en códigos de cálculo intensivo es posible

ejecutar de 500 a 3.500 operaciones aritméticas simultáneas. Nicolau y Fisher (1984) mostraron que un con programa Fortran estándar era posible la ejecución simultánea de 90 instrucciones para arquitecturas VLIW. Estos números muestran el lado optimista del paralelismo disponible.

David Wall (1991) indica que el límite del ILP (paralelismo a nivel de instrucción) es de alrededor de 5, raramente sobrepasando el 7. Bulter et al. (1991) indican que cuando se eliminan todas las restricciones el GDP puede exceder 17 instrucciones por ciclo. Si el hardware está perfectamente balanceado es posible conseguir de 2.0 a 5.8 instrucciones por ciclo en un procesador superescalar. Estos números muestran el lado pesimista del paralelismo disponible.

6.1.3. Rendimiento medio armónico. Ley de Amdahl

Consideremos un sistema paralelo con n procesadores ejecutando m programas en varios modos con diferentes niveles de rendimiento. Queremos definir el rendimiento medio de este tipo de multiprocesadores. Con una distribución de peso podemos definir una expresión del rendimiento.

Cada modo de ejecución puede corresponder a un tipo de ejecución como por ejemplo procesamiento escalar, vectorial, secuencial o paralela. Cada programa puede ejecutarse mediante una combinación de estos modos. El rendimiento *medio armónico* proporciona un rendimiento medio sobre la ejecución de un gran número de programas ejecutándose en varios modos.

Antes de obtener dicha expresión, estudiaremos las expresiones de la *media aritmética* y *geométrica* obtenidas por James Smith (1988). La velocidad de ejecución R_i para el programa i-ésimo se mide en MIPS o Mflops.

Media aritmética del rendimiento. Sea $\{R_i\}$ el conjunto de los rendimientos de los programas $i = 1, 2, \ldots, m$. La *media aritmética del rendimiento* se define como

$$R_a = \sum_{i=1}^{m} \frac{R_i}{m}$$

La expresión R_a supone que los m programas tienen el mismo peso $(1/m)$. Si existe una distribución de pesos de los distintos programas $\pi = \{f_i \text{ para } i = 1, 2, \ldots, m\}$, definimos la *media aritmética ponderada del rendimiento* como:

$$R_a^* = \sum_{i=1}^{m} (f_i R_i)$$

Esta media aritmética es proporcional a la suma de los inversos de los tiempos de ejecución; no es inversamente proporcional a la suma de los tiempos de ejecución. Por lo tanto, la media aritmética falla al representar el tiempo real consumido por los *benchmarks*.

Media geométrica del rendimiento. La *media geométrica de la velocidad de ejecución o rendimiento* para m programas se define como

$$R_g = \prod_{i=1}^{m} R_i^{1/m}$$

Con una distribución de pesos $\pi = \{f_i \text{ para } i = 1, 2, \ldots, m\}$, podemos definir una *media geométrica ponderada del rendimiento* como:

$$R_g^* = \prod_{i=1}^{m} R_i^{f_i}$$

La media geométrica tampoco capta el rendimiento real, ya que no presenta una relación inversa con el tiempo total. La media geométrica ha sido defendida para el uso con cifras de rendimiento que han sido normalizadas con respecto a una máquina de referencia con la que se está comparando.

Rendimiento medio armónico. Debido a los problemas que presentan la media aritmética y geométrica, necesitamos otra expresión del rendimiento medio basado en la media aritmética del tiempo de ejecución. De hecho, $T_i = 1/R_i$, es el tiempo medio de ejecución por instrucción para el programa i. La *media aritmética del tiempo de ejecución* por instrucción se define como

$$T_a = \frac{1}{m} \sum_{i=1}^{m} T_i = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{R_i}$$

La *media armónica de la velocidad de ejecución* sobre m programas de prueba se define por el hecho de que $R_h = \frac{1}{T_a}$:

$$R_h = \frac{m}{\sum_{i=1}^{m} \frac{1}{R_i}}$$

Con esto, el rendimiento medio armónico está de verdad relacionado con el tiempo medio de ejecución. Si consideramos una distribución de pesos, podemos definir el *rendimiento medio armónico ponderado* como:

$$R_h^* = \frac{1}{\sum_{i=1}^{m} \frac{f_i}{R_i}}$$

***Speed-up* armónico medio**. Otra forma de aplicar el concepto de media armónica es ligar los distintos modos de un programa con el número de procesadores usados. Supongamos que un programa (o una carga formada por la combinación de varios programas) se ejecutan en un sistema con n procesadores. Durante el periodo de ejecución, el programa puede usar $i = 1, 2, \ldots n$ procesadores en diferentes periodos de tiempo.

Decimos que el programa se ejecuta en *modo* i si usamos i procesadores. R_i se usa para reflejar la velocidad conjunta de i procesadores. Supongamos que $T_1 = 1/R_1 = 1$ es el tiempo de ejecución secuencial en un mono-procesador con una velocidad de ejecución $R_1 = 1$. Entonces $T_i = 1/R_i = 1/i$ es el tiempo de ejecución usando i procesadores con una velocidad de ejecución combinada de $R_i = i$ en el caso ideal.

Supongamos que un programa dado se ejecuta en n modos de ejecución con una distribución de pesos $w = \{f_i \text{ para } i = 1, 2, \ldots, n\}$. El *speed-up armónico medio ponderado* se define como:

$$S = T_1/T^* = \frac{1}{\left(\sum_{i=1}^{n} \frac{f_i}{R_i}\right)} \tag{6.3}$$

donde $T^* = 1/R_h^*$ es la *media armónica ponderada del tiempo de ejecución* para los n modos de ejecución.

La figura 6.2 muestra el comportamiento del *speed-up* para tres funciones de peso distintas.

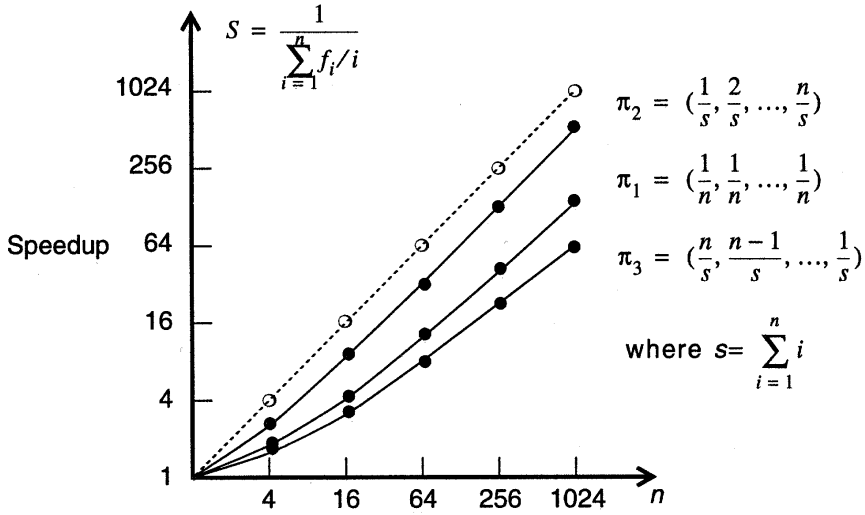


Figura 6.2: Media armónica del *speed-up* con respecto a tres distribuciones de probabilidad: π_1 para la distribución uniforme, π_2 en favor de usar más procesadores y π_3 en favor de usar menos procesadores.

Ley de Amdahl. De la expresión (6.3) de S se puede derivar la ley de Amdahl como sigue: En primer lugar supongamos que $R_i = i$ y $w = (\alpha, 0, 0, \ldots, 0, 1-\alpha)$. Esto implica que el sistema usa un modo secuencial puro con una probabilidad de α, o los n procesadores con una probabilidad de $1-\alpha$. Sustituyendo $R_1 = 1$, $R_n = n$ y w en la ecuación de S (6.3), obtenemos la siguiente expresión para el *speed-up*:

$$S_n = \frac{n}{1 + (n-1)\alpha} \tag{6.4}$$

A esta expresión se le conoce como la *ley de Amdahl*. La implicación es que $S \to 1/\alpha$ cuando $n \to \infty$. En otras palabras, independientemente del número de procesadores que se emplee, existe un límite superior del *speed-up* debido a la parte serie de todo programa.

En la figura 6.3 se han trazado las curvas correspondientes a la ecuación (6.4) para 4 valores de α. El *speed-up* ideal se obtiene para $\alpha = 0$, es decir, el caso en que no hay parte serie a ejecutar y todo el código es paralelizable. A poco que el valor de α sea no nulo, el *speed-up* máximo empieza a decaer muy deprisa.

Esta ley de Amdahl se puede generalizar y es aplicable a cualquier problema que tenga una parte *mejorable* y otra que no se pueda mejorar. Si llamamos F_m a la fracción del problema que se puede mejorar, la fracción de problema que no se puede mejorar será $(1 - F_m)$. Dado el problema se tiene una magnitud que es la que mejora con respecto a la inicial, a la magnitud inicial la podemos llamar M_{ini} y a la magnitud una vez aplicadas las mejoras M_{mej}. La mejora S_{up} es siempre el cociente entre las dos:

$$S_{up} = \frac{M_{ini}}{M_{mej}}$$

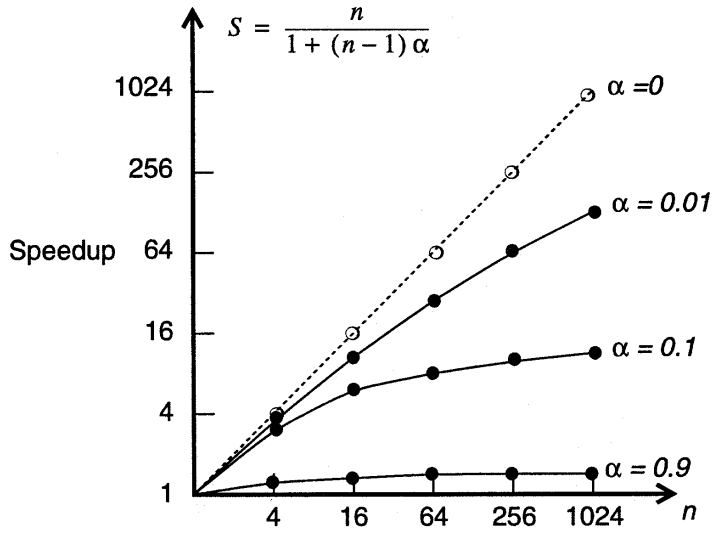


Figura 6.3: Mejora del rendimiento para diferentes valores de α, donde α es la fracción del cuello de botella secuencial.

Este cociente se puede poner en función de las fracciones que son mejorables y las que no, ya que $M_{ini} = K \cdot ((1 - F_m) + F_m) = K$, y $M_{mej} = K \cdot ((1 - F_m) + F_m/S_m)$, donde la K es una proporcionalidad con las unidades de la magnitud del problema, y S_m es el factor de mejora de la parte mejorable. Con todo esto se puede reescribir la mejora total del sistema al intentar mejorar S_m veces la parte mejorable como:

$$S_{up} = \frac{1}{1 - F_m + \frac{F_m}{S_m}}$$

que es la expresión de la ley de Amdahl generalizada que se puede aplicar a cualquier objeto que se quiera mejorar y sólo una parte sea mejorable. En el caso de los multiprocesadores el factor de mejora de la parte mejorable (paralelizable) es precisamente n, es decir, el número de procesadores. Por otro lado, la fracción que no es mejorable es la parte serie no paralelizable que llamábamos α. Con todo esto, sustituyendo los diferentes valores en la expresión anterior, y multiplicando por n/n, se tiene:

$$S_{up} = \frac{1}{\alpha + \frac{1-\alpha}{n}} = \frac{n}{1 + n\alpha - \alpha} = \frac{n}{1 + (n-1)\alpha}$$

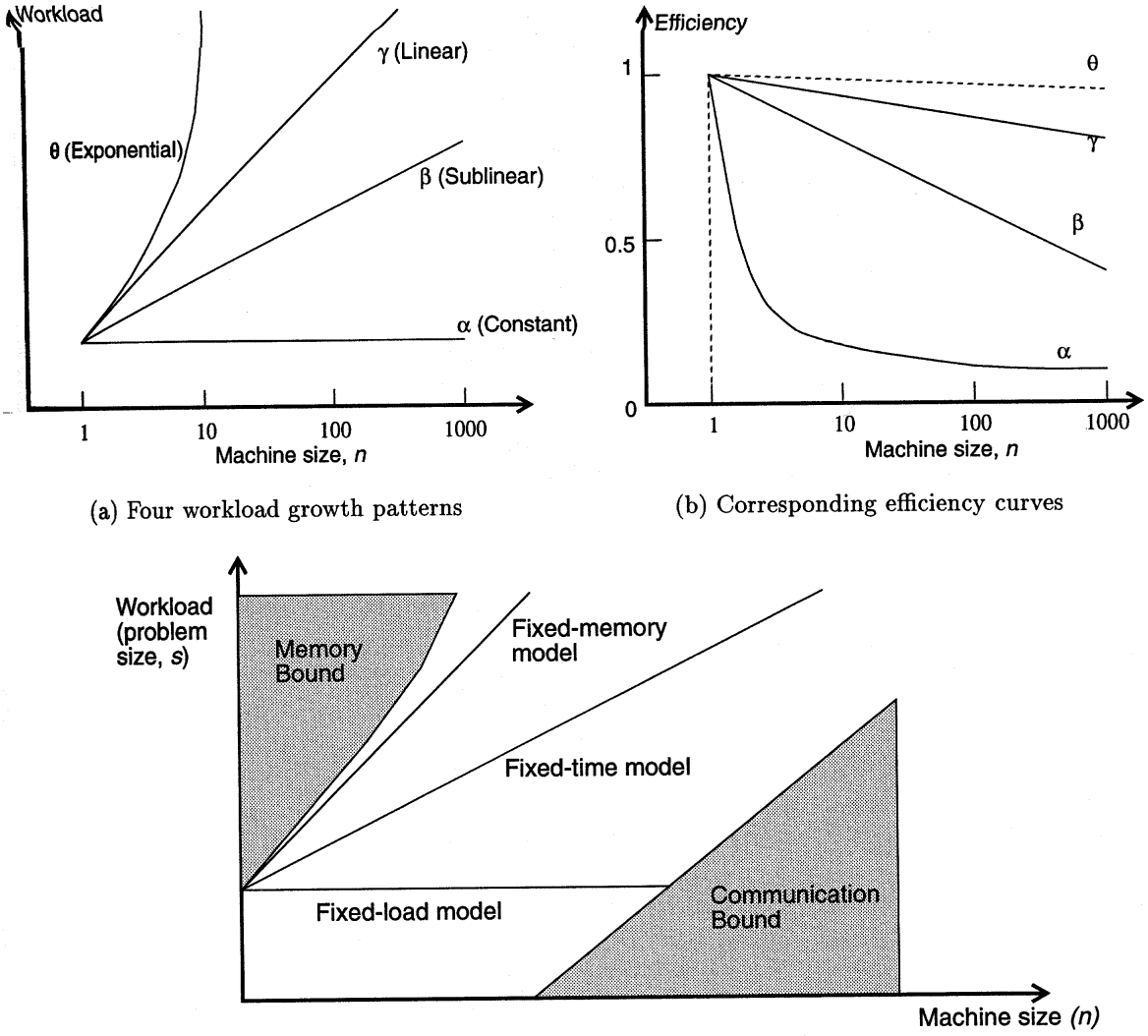
que es exactamente la misma expresión obtenida aplicando el *speed-up* medio armónico.

La expresión de la ley de Amdahl generalizada se puede aplicar a cualquier problema. Por ejemplo, el rendimiento relativo vectorial/escalar en los procesadores vectoriales, no es más que la aplicación de esta ley al caso en que un programa tenga una parte vectorizable (parte que va a mejorar) y otra escalar, cuyo rendimiento no va a mejorar por el hecho de estar utilizando un procesador vectorial.

6.2. Modelos del rendimiento del *speed-up*

En esta sección se describen tres modelos de medición del *speed-up*. La ley de Amdahl (1967) se basa una carga de trabajo fija o en un problema de tamaño fijo. La ley de

Gustafson (1987) se aplica a problemas escalables, donde el tamaño del problema se incrementa al aumentar el tamaño de la máquina o se dispone de un tiempo fijo para realizar una determinada tarea. El modelo de *speed-up* de Sun y Ni (1993) se aplica a problemas escalables limitados por la capacidad de la memoria. En la figura 6.4 se muestra un esquema de los tres modelos utilizados.



(a) Four workload growth patterns

(b) Corresponding efficiency curves

Figura 6.4: Modelos de rendimiento del *speed-up*.

6.2.1. Ley de Amdahl, limitación por carga de trabajo fija

En muchas aplicaciones prácticas, donde es importante la respuesta más rápida posible, la carga de trabajo se mantiene fija y es el tiempo de ejecución lo que se debe intentar reducir. Al incrementarse el número de procesadores en el sistema paralelo, la carga fija se distribuye entre más procesadores para la ejecución paralela. Por lo tanto, el objetivo principal es obtener los resultados lo más pronto posible. En otras palabras, disminuir el tiempo de respuesta es nuestra principal meta. A la ganancia de tiempo obtenida para este tipo de aplicaciones donde el tiempo de ejecución es crítico se le denomina *speed-up bajo carga fija.*

***Speed-up* bajo carga fija**. La fórmula vista en el apartado anterior se basa en una carga de trabajo fija, sin importar el tamaño de la máquina. Las formulaciones tradicionales del *speed-up*, incluyendo la ley de Amdahl, están basadas en un problema de tamaño fijo y por lo tanto en una carga fija. En este caso, el factor de *speed-up* está acotado superiormente por el cuello de botella secuencial.

A continuación se consideran las dos posibles situaciones: GDP$< n$ ó GDP$\geq n$. Consideremos el caso donde el GDP$= i \geq n$. Supongamos que todos los n procesadores se usan para ejecutar W_i exclusivamente. El tiempo de ejecución de W_i es

$$t_i(n) = \frac{W_i}{i\Delta} \left\lceil \frac{i}{n} \right\rceil$$

De esta manera el tiempo de respuesta es

$$T(n) = \sum_{i=1}^{m} \frac{W_i}{i\Delta} \left\lceil \frac{i}{n} \right\rceil$$

Observar que si $i < n$, entonces $t_i(n) = t_i(\infty) = W_i/i\Delta$. Ahora, definimos el *speed-up para carga fija* como :

$$S_n = \frac{T(1)}{T(n)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i} \left\lceil \frac{i}{n} \right\rceil}$$

Observar que $S_n \leq S_\infty \leq A$.

Existe una gran cantidad de factores que se han ignorado que pueden rebajar el *speed-up*. Estos factores incluyen latencias de comunicaciones debidas a retrasos en el acceso a la memoria, comunicaciones a través de un bus o red, o sobrecarga del sistema operativo y retrasos causados por las interrupciones. Si $Q(n)$ es la suma de todas las sobrecargas del sistema en un sistema con n procesadores, entonces:

$$S_n = \frac{T(1)}{T(n) + Q(n)} = \frac{\sum_{i=1}^{m} W_i}{\sum_{i=1}^{m} \frac{W_i}{i} \left\lceil \frac{i}{n} \right\rceil + Q(n)} \tag{6.5}$$

El retraso por sobrecarga $Q(n)$ depende siempre de la aplicación y de la máquina. Es muy difícil obtener una expresión para $Q(n)$. A no ser de que se especifique otra cosa se supondrá que $Q(n) = 0$ para simplificar la explicación.

Ley de Amdahl revisada. En 1967, Gene Amdahl derivó un *speed-up* para el caso particular donde el computador opera en modo puramente secuencial (GDP$= 1$) o en modo totalmente paralelo (GDP$= n$). Es decir, $W_i = 0$ si $i \neq 1$ ó $i \neq n$. En este caso, el *speed-up* viene dado por:

$$S_n = \frac{W_1 + W_n}{W_1 + W_n/n} \tag{6.6}$$

La ley de Amdahl supone que la parte secuencia del programa W_i no cambia con respecto a tamaño n de la máquina. Sin embargo, la porción paralela se ejecuta equitativamente por los n procesadores reduciéndose el tiempo de ejecución para esta parte.

Suponiendo una situación normalizada en la cual $W_1 = \alpha$ y $W_n = 1 - \alpha$ se tiene que $W_1 + W_n = \alpha + 1 - \alpha = 1$. Con esta sustitución, la ecuación (6.4) y (6.6) son la

misma. Igual que en aquella expresión α es la fracción serie del programa y $(1-\alpha)$ la paralelizable.

La ley de Amdahl se ilustra en la figura 6.5. Cuando el número de procesadores aumenta, la carga ejecutada en cada procesador decrece. Sin embargo, la cantidad total de trabajo (carga) $W_1 + W_n$ se mantiene constante como se muestra en la figura 6.5a. En la figura 6.5b, el tiempo total de ejecución decrece porque $T_n = W_n/n$. Finalmente, el término secuencial domina el rendimiento porque $T_n \to 0$ al hacer n muy grande siendo T_1 constante.

Cuello de botella secuencial. La figura 6.5c muestra una gráfica de la ley de Amdahl para diferentes valores de $0 \leq \alpha \leq 1$. El máximo *speed-up*, $S_n = n$, se obtiene para $\alpha = 0$. El mínimo *speed-up*, $S_n = 1$, se obtiene para $\alpha = 1$. Cuando $n \to \infty$, el valor límite es $S_\infty = 1/\alpha$. Esto implica que el *speed-up* está acotado superiormente por $1/\alpha$, independientemente del tamaño de la máquina.

La curva del *speed-up* en la figura 6.5c cae rápidamente al aumentar α. Esto significa que con un pequeño porcentaje de código secuencial, el rendimiento total no puede ser superior a $1/\alpha$. A este α se le denomina *cuello de botella secuencial* de un programa.

El problema de un cuello de botella secuencial no puede resolverse incrementando el número de procesadores del sistema. El problema real está en la existencia de una fracción secuencial (s) del código. Esta propiedad ha impuesto una visión muy pesimista del procesamiento paralelo en las pasadas dos décadas.

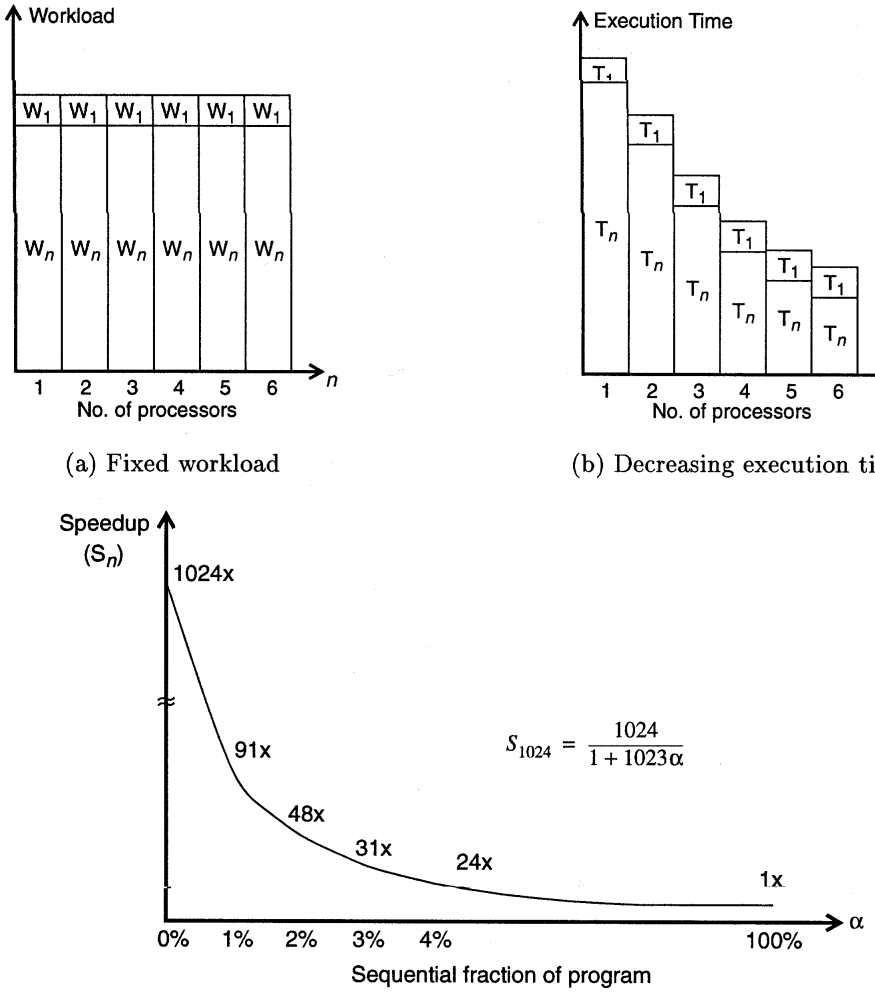
De hecho, se observaron dos impactos de esta ley en la industria de los computadores paralelos. En primer lugar, los fabricantes dejaron de lado la construcción de computadores paralelos de gran escala. En segundo lugar, una parte del esfuerzo investigador se desplazó al campo de desarrollo de compiladores paralelos en un intento de reducir el valor de α y mejorar de esa forma el rendimiento.

6.2.2. Ley de Gustafson, limitación por tiempo fijo

Uno de los mayores inconvenientes de aplicar la ley de Amdahl es que el problema (la carga de trabajo) no puede aumentarse para corresponderse con el poder de cómputo al aumentar el tamaño de la máquina. En otras palabras, el tamaño fijo impide el escalado del rendimiento. Aunque el cuello de botella secuencial es un problema importante, puede aliviarse en gran medida eliminando la restricción de la carga fija (o tamaño fijo del problema). John Gustafson (1988) ha propuesto el concepto de tiempo fijo que da lugar a un modelo del *speed-up* escalado.

Escalado para conseguir una mayor precisión. Las aplicaciones de tiempo-real son la causa principal del desarrollo de un modelo de *speed-up* de carga fija y la ley de Amdahl. Existen muchas otras aplicaciones que enfatizan la precisión más que el tiempo de respuesta. Al aumentar el tamaño de la máquina para obtener mayor potencia de cálculo, queremos incrementar el tamaño del problema para obtener una mayor carga de trabajo, produciendo una solución más precisa y manteniendo el tiempo de ejecución.

Un ejemplo de este tipo de problemas es el cálculo de la predicción meteorológica. Habitualmente se tiene un tiempo fijo para calcular el tiempo que hará en unas horas, naturalmente se debe realizar el cálculo antes de que la lluvia llegue. Normalmente se suele imponer un tiempo fijo de unos 45 minutos a una hora. En ese tiempo se tiene que obtener la mayor precisión posible. Para calcular la predicción se sigue un modelo físico que divide el área terrestre a analizar en cuadrados, de manera que los cálculos



(a) Fixed workload

(b) Decreasing execution time

(c) Speedup with a fixed load

Figura 6.5: Modelo del *speed-up* de carga fija y la ley de Amdahl.

realizados en cada uno de estos cuadrados se puede hacer en paralelo. Si se disponen de muchos procesadores se podrán hacer cuadros más pequeños con lo que la precisión aumenta manteniendo el tiempo de ejecución.

***Speed-up* de tiempo fijo**. En aplicaciones de precisión crítica, se desea resolver un problema de mayor tamaño en una máquina mayor con aproximadamente el mismo tiempo de ejecución que costaría resolver un problema menor en una máquina menor. Al aumentar el tamaño de la máquina, tendremos una nueva carga de trabajo y por lo tanto un nuevo perfil del paralelismo. Sea m' el máximo GDP con respecto al problema escalado y W_i' la carga de trabajo con GDP$= i$.

Observar que, en general, $W_i' > W_i$ para $2 \leq i \leq m'$ y $W_1' = W_1$. El *speed-up* de tiempo fijo se define bajo el supuesto de que $T(1) = T'(n)$, donde $T'(n)$ es el tiempo de ejecución del problema escalado y $T(1)$ se corresponde con el problema original sin

escalar. Así, tenemos que

$$\sum_{i=1}^{m} W_i = \sum_{i=1}^{m'} \frac{W'_i}{i} \left\lceil \frac{i}{n} \right\rceil + Q(n)$$

Una fórmula general para el *speed-up* de tiempo fijo se define por $S'_n = T'(1)/T'(n) = T'(1)/T(1)$. Por analogía con la ecuación (6.5) se obtiene la expresión para el *speed-up* de tiempo fijo:

$$S'_n = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m} \frac{W'_i}{i} \left\lceil \frac{i}{n} \right\rceil + Q(n)} = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m} W_i} \tag{6.7}$$

Ley de Gustafson. El *speed-up* de tiempo fijo fue desarrollado por Gustafson para un perfil de paralelismo especial con $W_i = 0$ si $i \neq 1$ y $i \neq n$. De forma similar a la ley de Amdahl, podemos reescribir la ecuación anterior (6.7) como sigue (suponemos $Q(n) = 0$):

$$S'_n = \frac{\sum_{i=1}^{m'} W'_i}{\sum_{i=1}^{m} W_i} = \frac{W'_1 + W'_n}{W_1 + W_n} = \frac{W_1 + nW_n}{W_1 + W_n}$$

La figura 6.6a muestra la relación del escalado de la carga de trabajo con el *speed-up* escalado de Gustafson. De hecho, la ley de Gustafson puede reformularse en términos de $\alpha = W_1$ y $1 - \alpha = W_n$, bajo la suposición de que $W_1 + W_n = 1$, como sigue:

$$S'_n = \frac{\alpha + n(1-\alpha)}{\alpha + (1-\alpha)} = n - \alpha(n-1)$$

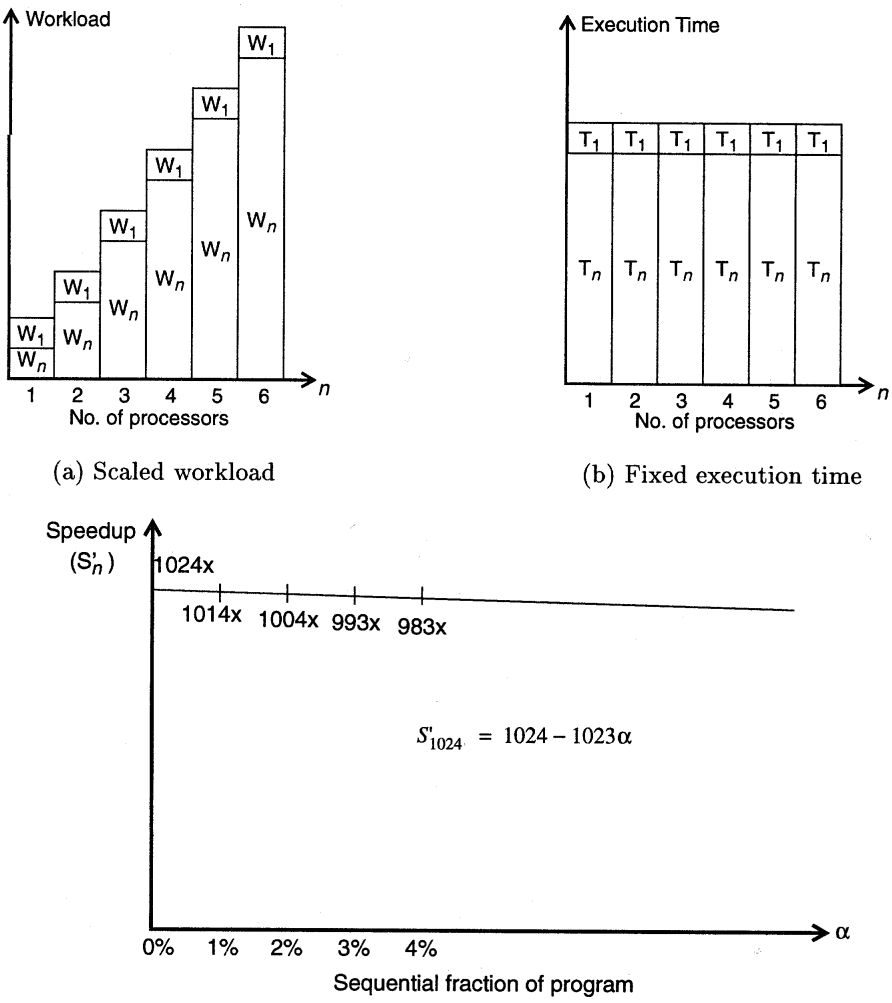
Obsérvese que la pendiente de la curva S_n en la figura 6.6c es mucho más plana que en la figura 6.5c. Esto implica que la ley de Gustafson soporta el rendimiento escalable al aumentar el tamaño de la máquina. La idea es mantener a todos los procesadores ocupados incrementando el tamaño del problema.

6.2.3. Modelo del *speed-up* limitado por la memoria fija

Xian-He Sun y Lionel Ni (1993) han desarrollado un modelo del *speed-up* limitado por la memoria que generaliza la ley de Amdahl y Gustafson para maximizar el uso de la CPU y la memoria. La idea es resolver el mayor problema posible, limitado por el espacio de memoria. En este caso también es necesario una carga de trabajo escalada, proporcionando un mayor *speed-up*, mayor precisión y mejor utilización de los recursos.

Problemas limitados por el espacio de memoria. Los cálculos científicos y las aplicaciones de ingeniería suelen necesitar una gran cantidad de memoria. De hecho, muchas aplicaciones de los ordenadores paralelos surgen de la limitación de la memoria más que de la CPU o la E/S. Esto es especialmente cierto en sistemas multicomputador con memoria distribuida. Cada elemento de proceso está limitado a usar su propia memoria local por lo que sólo puede hacer frente a un pequeño subproblema.

Cuando se utiliza un mayor número de nodos para resolver un problema grande, la capacidad de memoria total se incrementa de forma proporcional. Esto le permite al sistema resolver un problema escalado mediante el particionamiento del programa y la descomposición del conjunto de datos.



(a) Scaled workload

(b) Fixed execution time

(c) Speedup with fixed execution time

Figura 6.6: Modelo de *speed-up* de tiempo fijo y la ley de Gustafson.

En lugar de mantener fijo el tiempo de ejecución, uno puede querer usar toda la memoria disponible para aumentar aún más el tamaño del problema. En otras palabras, si se tiene un espacio de memoria adecuado y el problema escalado cumple el límite de tiempo impuesto por la ley de Gustafson, se puede incrementar el tamaño del problema, consiguiendo una mejor solución o una solución más precisa.

El modelo de limitación de la memoria se desarrolló bajo esta filosofía. La idea es resolver el mayor problema posible, limitado únicamente por la capacidad de memoria disponible.

***Speed-up* de memoria fija**. Sea M el requisito de memoria para un problema dado y W la carga computacional. Ambos factores están relacionados de varias formas, dependiendo del direccionamiento del espacio y las restricciones de la arquitectura. Así, podemos escribir $W = g(M)$ o $M = g^{-1}(W)$.

En un multicomputador, y en la mayoría de multiprocesadores, la capacidad total de la memoria se incrementa linealmente con el número de nodos disponibles. Sea $W = \sum_{i=1}^{m} W_i$ la carga para una ejecución secuencial del programa en un único nodo, y

$W^* = \sum_{i=1}^{m^*} W_i^*$ la carga para el problema para n nodos, donde m^* es el máximo GDP del problema escalado. Los requisitos de memoria para un nodo activo está limitado por $M = g^{-1}\left(\sum_{i=1}^{m} W_i\right)$.

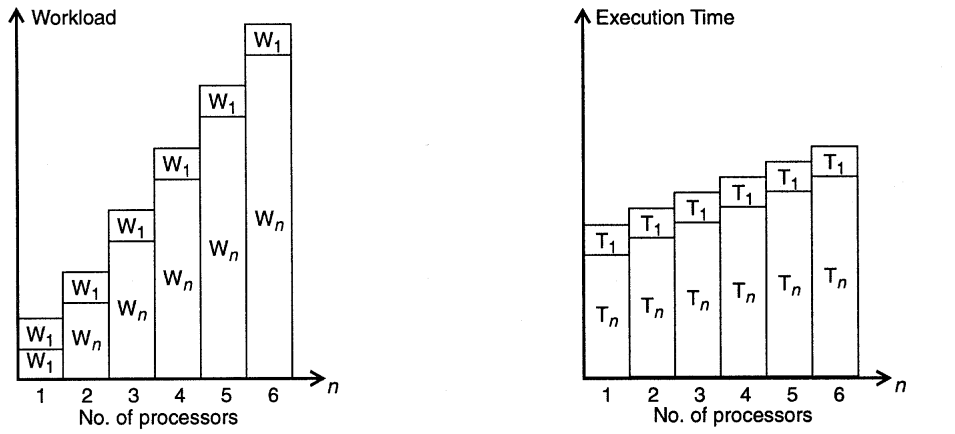
El *speed-up con memoria fija* se define de forma similar al caso de la ecuación (6.7):

$$S_n^* = \frac{\sum_{i=1}^{m^*} W_i^*}{\sum_{i=1}^{m^*} \frac{W_i^*}{i} \left\lceil \frac{i}{n} \right\rceil + Q(n)} \tag{6.8}$$

La carga de trabajo para la ejecución secuencial en un único procesador es independiente del tamaño del problema o del tamaño del sistema. Así, podemos escribir $W_1 = W'_1 = W_1^*$ para los tres modelos de *speed-up*. Consideremos el caso especial con dos modos de operación: ejecución *secuencial* frente a *perfectamente paralela*. La mejora en la memoria está relacionada con la carga escalada mediante la fórmula $W_n^* = g^*(nM)$, donde nM es el incremento en la capacidad de la memoria para un multicomputador con n nodos.

Supongamos además que $g^*(nM) = G(n)g(M) = G(n)W_n$, donde $W_n = g(M)$ y g^* es una función homogénea. El factor $G(n)$ refleja el incremento en la carga al aumentar la memoria n veces. Esto nos permite rescribir la fórmula anterior bajo la suposición de que $W_i = 0$ si $i \neq 1$ o n y $Q(n) = 0$:

$$S_n^* = \frac{W_1^* + W_n^*}{W_1^* + W_n^*/n} = \frac{W_1 + G(n)W_n}{W_1 + G(n)W_n/n} \tag{6.9}$$



(a) Scaled workload (b) Slightly increased execution time

Figura 6.7: Modelo de *speed-up* de memoria fija.

Rigurosamente hablando este modelo sólo es válido bajo estas dos suposiciones: (1) El conjunto de toda la memoria forma un espacio global de direcciones (en otras palabras, suponemos un espacio de memoria compartido distribuido); (2) Todo el espacio de memoria disponible se utiliza para el problema escalado. Existen tres casos especiales donde se puede aplicar la ecuación (6.9):

1. $G(n) = 1$. Se corresponde con el caso donde el tamaño del problema es fijo, siendo equivalente a la ley de Amdahl.

2. $G(n) = n$. Se aplica al caso en el que la carga se incrementa n veces cuando la memoria se incrementa n veces. En este caso, la ecuación se corresponde con la ley de Gustafson con un tiempo de ejecución fijo.
3. $G(n) > n$. Se corresponde con la situación donde la carga computacional se incrementa más rápidamente que los requisitos de memoria. En este caso, el modelo de memoria fija da posiblemente todavía un mayor *speed-up* que el de tiempo fijo.

De este análisis se pueden obtener las siguientes conclusiones: la ley de Amdahl y la de Gustafson son casos particulares del modelo de tiempo fijo. Cuando la computación crece más rápidamente que los requisitos de memoria, lo que es frecuente en el caso de algunas simulaciones científicas y aplicaciones de ingeniería, el modelo de memoria fija (figura 6.7) da lugar a un mayor *speed-up* (es decir, $S_n^* \geq S'_n \geq S_n$) y una mejor utilización de los recursos.

6.3. Modelos del rendimiento según la granularidad

El contenido de esta sección y resto del capítulo se encuentran en el libro [Sto93].

Un parámetro que se suele dar para caracterizar los sistemas multiprocesadores es el **rendimiento de pico** o rendimiento máximo del sistema. Habitualmente este rendimiento de pico se suele calcular como el número de procesadores del sistema multiplicado por el rendimiento de cada uno de los procesadores.

Cuando el sistema opera al rendimiento máximo todos los procesadores están realizando un trabajo útil; ningún procesador está parado y ningún procesador ejecuta alguna instrucción extra que no estuviera en el algoritmo original. En este estado de rendimiento de pico todos los n procesadores están contribuyendo al rendimiento efectivo del sistema y la velocidad de procesamiento viene incrementada por un factor n.

El estado de rendimiento máximo o de pico es un estado raro que difícilmente se puede alcanzar. Hay varios factores que introducen ineficiencia. Algunos de estos factores son los siguientes:

- Retrasos introducidos por las comunicaciones entre procesos.
- La sobrecarga de trabajo debida a la necesidad de sincronizar el trabajo entre los distintos procesadores.
- La pérdida de eficiencia cuando algún procesador se queda sin trabajo para realizar.
- La pérdida de eficiencia cuando uno o más procesadores realizan algún esfuerzo inútil.
- El coste de procesamiento para controlar el sistema y la programación de operaciones.

Estos problemas se hacen realmente serios cuando el número de procesadores es elevado, es decir, es difícil mantener un bajo grado de ineficiencia al aumentar el número de procesadores. Normalmente se obtiene una eficiencia bastante alta con sistemas con pocos procesadores (4-16) pero esta eficiencia se ve seriamente reducida cuando el número de procesadores es alto. Dar el rendimiento de pico de un multiprocesador con pocos procesadores puede dar una idea de su rendimiento efectivo, pero el rendimiento de pico en un multiprocesador con muchos procesadores sólo debe considerarse como parte de las especificaciones, ya que no tiene por qué dar una estimación real del rendimiento del sistema.

A continuación se pretende estudiar la influencia de la sobrecarga de procesamiento por el hecho de añadir más procesadores al cálculo. Se va a comprobar que el rendimiento de un sistema multiprocesador depende fuertemente de la relación R/C, donde R es una unidad de ejecución (con unidades de tiempo o instrucciones por segundo), y C es la sobrecarga debida a las comunicaciones producidas por R. El cociente de los dos da la cantidad de sobrecarga que aparece por unidad de cómputo. Cuando la relación es pequeña no resulta provechoso paralelizar porque aparece mucha sobrecarga. Cuando la relación da un número muy alto entonces es beneficioso paralelizar puesto que la sobrecarga que aparece es pequeña. Normalmente el factor R/C da un valor alto siempre que se divida el problema en trozos grandes, ya que entonces las comunicaciones serán pequeñas comparativamente.

El factor R/C da también idea de la granularidad del sistema, es decir, de lo mucho que se ha dividido el problema en pedazos:

Grano grueso: Un sistema cuyo paralelismo es de *grano grueso* suele tener un factor R/C relativamente grande puesto que los trozos R son grandes y producen un coste de comunicaciones relativamente pequeño. Si un sistema es de grano grueso es beneficioso paralelizar puesto que R/C es grande, pero si los trozos en que se divide el problema son grandes, el problema queda dividido en pocos trozos y el rendimiento máximo no es muy alto (pocas unidades funcionando en paralelo).

Grano fino: Un sistema cuyo paralelismo es de grano fino suele tener un factor R/C pequeño puesto que los trozos R en que se ha dividido el problema son pequeños. Normalmente, si se divide el problema en trozos muy pequeños se obtiene un R/C pequeño por lo que no resulta muy beneficioso paralelizar, pero al ser los trozos pequeños el problema puede quedar muy dividido y cada unidad funcional puede realizar una tarea distinta. En estos casos se podría alcanzar un gran rendimiento por el gran paralelismo existente, pero no se alcanza puesto que el factor R/C es pequeño.

En resumen: si se tiene un problema muy paralelizable (divisible en muchos pedazos) normalmente no va a ser interesante paralelizar tanto puesto que las sobrecargas no van a permitir aprovechar todo ese rendimiento potencial. Si un problema es poco paralelizable (divisible en pocos pedazos) en rendimiento máximo alcanzable es pequeño, ya que se ha dividido el problema en pocos trozos que se ejecutarán en paralelo, pero el paralelismo obtenido será bastante eficiente puesto que las comunicaciones entre trozos grandes son escasas.

6.3.1. Modelo básico: 2 procesadores y comunicaciones no solapadas

Vamos a suponer que cierta aplicación tiene M tareas a realizar las cuales se pueden llevar a cabo de forma paralela. El objetivo está en ejecutar estas M tareas en un sistema con N procesadores en el menor tiempo posible. Para empezar el análisis se comenzará con $N = 2$ procesadores y luego se extenderá el razonamiento a cualquier número de procesadores.

Como punto de partida realizaremos las siguientes suposiciones (más tarde se pueden relajar para obtener resultados más realistas):

1. Cada tarea se ejecuta en R unidades de tiempo.
2. Cada tarea de un procesador se comunica con todas las tareas del resto de proce-

sadores con un coste de sobrecarga de C unidades de tiempo. El coste de comunicaciones con las tareas en el mismo procesador es cero.

Si se tienen dos procesadores se pueden repartir las tareas entre ellos. En un caso extremo se le pueden dar todas las tareas a un único procesador, y en el otro caso extremo se puede realizar un reparto igualitario de tareas entre los dos procesadores. Entre estos dos extremos se tienen situaciones donde un procesador tiene k tareas mientras que el otro tiene $(M - k)$ donde k puede ser cualquier reparto entre 0 y M. En cualquier caso el tiempo de ejecución va a tener dos términos, uno debido al coste de ejecución de las tareas (función de R) y otro debido a la sobrecarga por las comunicaciones (función de C). La expresión que se obtiene para el tiempo de ejecución (T_e) es la siguiente:

$$T_e = R \text{ máx}\{M - k, k\} + C(M - k)k \tag{6.10}$$

El tiempo propio de ejecución de las tareas es el término R máx$(M - k, k)$ y es lineal con k. El término debido a la sobrecarga es $C(M - k)k$ y es un término que crece cuadráticamente con k. Cuanto más repartidas se encuentran las tareas menor es el término debido a R y mayor es el debido a C. Esto significa que, o bien el tiempo mínimo se obtiene cuando las tareas están igualitariamente repartidas, o bien cuando sólo un procesador ejecuta todas las tareas, no hay término medio.

Las contribuciones de R y C al tiempo de ejecución total se pueden ver mejor en la figura 6.8. En la figura 6.8(a) se muestra la situación en la cual el coste de las comunicaciones es tan alto que resulta más provechoso ejecutar todas las tareas en un único procesador. En la figura 6.8(b) se da la situación en la cual el coste de las comunicaciones es menor que la ganancia obtenida por el hecho de paralelizar, por lo que en este caso es rentable dividir las tareas entre los dos procesadores.

Para obtener la relación R/C a partir de la cual es beneficioso paralelizar basta con igualar el tiempo de ejecución con reparto igualitario ($k = M/2$ reparto igualitario) con el tiempo de ejecución con un único procesador (RM) e igualar los términos debidos a R y C, es decir:

$$RM = R\frac{M}{2} + C\frac{M}{2}\frac{M}{2}$$

$$R\frac{M}{2} = C\frac{M}{2}\frac{M}{2}$$

realizando unas operaciones básicas de sustitución se tiene finalmente una cota para R/C:

$$\frac{R}{C} = \frac{M}{2} \tag{6.11}$$

esto quiere decir que si $R/C > M/2$ entonces resulta beneficioso paralelizar, y en caso contrario resulta mejor dejar todas las tareas en un único procesador.

6.3.2. Extensión a N procesadores

En el caso de tener N procesadores se puede suponer que cada uno ejecuta k_i tareas, de manera que $M = \sum_{i=1}^{N} k_i$. Con esto se puede generalizar la ecuación 6.10 obteniéndo-

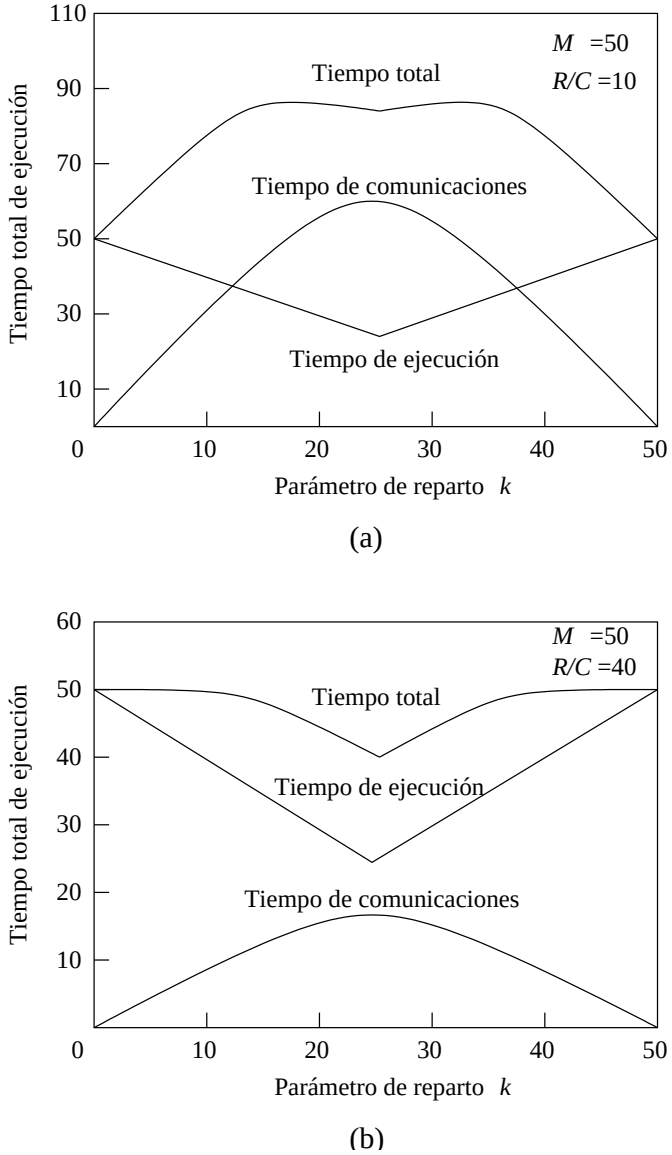


Figura 6.8: Tiempo de ejecución para dos factores R/C diferentes.

se la siguiente expresión para el tiempo total de ejecución con N procesadores:

$$\begin{aligned} T_e &= R\ \text{máx}\{k_i\} + \tfrac{C}{2} \textstyle\sum_{i=1}^{N} k_i(M - k_i) \\ &= R\ \text{máx}\{k_i\} + \tfrac{C}{2} \left(M^2 - \textstyle\sum_{i=1}^{N} k_i^2\right) \end{aligned} \tag{6.12}$$

Al igual que ocurría en el caso anterior, el mínimo de esta función se obtiene, o bien cuando sólo un procesador tiene todas las tareas, o bien, cuando se reparten de forma igualitaria las tareas. En este último caso el reparto igualitario es un tanto especial, ya que se deben repartir dándole a cada procesador un número $\left\lceil \frac{M}{N} \right\rceil$ de tareas hasta que queden menos de $\left\lceil \frac{M}{N} \right\rceil$ tareas que se le asignan a uno de los procesadores que queden; esto significa que pueden haber algunos procesadores que no reciban ninguna tarea.

Por lo tanto, el reparto igualitario deja a p procesadores con $\left\lceil \frac{M}{N} \right\rceil$ tareas, a otro se le asignan $M - \left\lceil \frac{M}{N} \right\rceil p$ tareas, y el resto de procesadores no tiene ninguna. Se puede

demostrar que este reparto, y no otro, es el que da el mínimo. La demostración es como sigue: Supongamos que $k_1 = \lceil \frac{M}{N} \rceil$ es el máximo número de tareas que tiene un procesador. Con el reparto propuesto sólo puede haber un procesador con menos de estas tareas a menos que no tenga ninguna. Vamos a suponer que hay dos procesadores que tienen menos tareas, en vez de uno, y veremos cómo al aumentar las tareas de uno de ellos se reduce el tiempo total de ejecución.

En efecto, sean k_2 y k_3 las tareas asignadas a dos procesadores que cumplen que $k_1 > k_2 \geq k_3 \geq 1$. Supongamos a continuación que pasamos una tarea del procesador que tiene k_3 al procesador que tiene k_2. El coste de ejecución debido a R no cambia, ya que el máximo de tareas sigue siendo el mismo. Sin embargo el tiempo de ejecución debido a las comunicaciones varía. En efecto, inicialmente el término exacto que varía es:

$$\frac{C}{2}\left(M^2 - (k_1^2 + k_2^2 + k_3^2 + \ldots)\right)$$

si ahora se hace $k_2 = k_2 + 1$ y $k_3 = k_3 - 1$, el término anterior pasa a ser:

$$\begin{array}{rl} = & \frac{C}{2}\left(M^2 - (k_1^2 + (k_2+1)^2 + (k_3-1)^2 + \ldots)\right) \\ = & \frac{C}{2}\left(M^2 - (k_1^2 + k_2^2 + 1 + 2k_2 + k_3^2 + 1 - 2k_3 + \ldots)\right) \\ = & \frac{C}{2}\left(M^2 - (k_1^2 + k_2^2 + k_3^2 + \ldots)\right) - \frac{C}{2}(2 + 2k_2 - 2k_3) \\ < & \frac{C}{2}\left(M^2 - (k_1^2 + k_2^2 + k_3^2 + \ldots)\right) \end{array}$$

es decir, al pasar una tarea de un procesador que tiene unas tareas k_3 a otro procesador que tiene las mismas o más tareas k_2 pero sin llegar al máximo, se reduce el tiempo de ejecución en un factor $(C/2)(2 + 2k_2 - 2k_3)$, que como $k_2 \geq k_3$ siempre será mayor que cero.

El **umbral** del factor R/C a partir del cual resulta interesante el reparto de tareas coincide con el visto para los dos procesadores y es $R/C = M/2$. Para hacer la demostración para N procesadores cualesquiera basta con igualar el tiempo de ejecución con un procesador (RM) y el tiempo de ejecución con los N procesadores:

$$RM = \frac{RM}{N} + \frac{CM^2}{2} - \frac{CM^2}{2N}$$

$$R\frac{N-1}{N} = C\frac{M}{2}\left(1 - \frac{1}{N}\right)$$

$$\frac{R}{C}\left(1 - \frac{1}{N}\right) = \frac{M}{2}\left(1 - \frac{1}{N}\right)$$

$$\frac{R}{C} = \frac{M}{2}$$

Resulta interesante calcular el ***speed-up*** para ver cómo influye el factor R/C en la mejora del rendimiento por el hecho de añadir procesadores al sistema. El *speed-up* es

siempre la relación entre el tiempo de ejecución con un procesador y con muchos:

$$\begin{aligned} Speedup &= \frac{RM}{\frac{RM}{N} + \frac{CM^2}{2} - \frac{CM^2}{2N}} \\ &= \frac{R}{\frac{R}{N} + \frac{CM(1-1/N)}{2}} \\ &= \frac{\frac{R}{C}N}{\frac{R}{C} + \frac{M(N-1)}{2}} \end{aligned}$$

Si $\frac{R}{C} \gg \frac{M(N-1)}{2}$ entonces $Speedup \approx N$, es decir, si el tiempo debido a la sobrecarga es muy pequeño comparado con el coste de ejecución entonces el sistema es bastante eficiente puesto que el *speed-up* crece con el número de procesadores. Llega un momento en que no importa lo grande que sea el factor R/C, ya que siempre hay un número de procesadores a partir del cual este factor ya no se puede considerar pequeño y el *speed-up* deja de crecer linealmente con N.

Al igual que ocurre con la ley de Amdahl llega un momento en que añadir más procesadores aumenta muy poco el rendimiento llegándose a un límite de mejora que no se puede superar. En efecto, haciendo el límite para $N \to \infty$, se llega a que la asíntota para el *speed-up* es:

$$S_{asint} = 2\frac{R}{CM}$$

Como resumen se puede decir que la sobrecarga debida a las comunicaciones juega un gran papel en la mejora del rendimiento en un sistema. No importa el rendimiento de pico que pueda tener un sistema; si la sobrecarga debida a las comunicaciones es relativamente alta, bastarán unos pocos procesadores para obtener mejor rendimiento que con muchos procesadores. En esto, la correcta división del software, es decir, la granularidad de la aplicación, juega también un papel importante en el rendimiento final del sistema.

Tareas no uniformes

Al principio de esta sección se hacía la suposición de que todas las M tareas se ejecutaban en el mismo tiempo R. El caso habitual es que cada tarea tenga su propio tiempo de ejecución lo que puede complicar bastante el análisis del reparto de tareas.

La forma de repartir las tareas cuando cada una tiene un tiempo de ejecución diferente sería la siguiente:

1. El término debido a R se minimiza siempre que se tenga un reparto igualitario, pero la igualdad se refiere al tiempo y no al número de tareas, por lo tanto se intentará repartir las tareas de manera que el tiempo de ejecución de todos los procesadores sea el mismo. Esto implicará que algunos procesadores tengan más tareas que otros.

2. El término debido a las comunicaciones se puede minimizar realizando un reparto lo más desparejo posible. Esto significa que manteniendo la regla anterior hay que intentar agrupar las tareas de manera que unos procesadores tengan muchas y otros procesadores tengan muy pocas.
3. Las dos reglas anteriores no aseguran que se vaya a obtener un tiempo de ejecución mínimo por lo que habrá que revisar el reparto obtenido.

Esta forma de repartir tareas entre procesadores no asegura la obtención del tiempo mínimo de ejecución aunque puede llegar a acercarse bastante. Existen métodos estadísticos para obtener de forma segura repartos mejores.

6.3.3. Otras suposiciones para las comunicaciones

En la sección anterior se había supuesto que unas tareas en unos procesadores se comunicaban con otras tareas en otros procesadores y viceversa, lo que provocaba la aparición de un término debido a las comunicaciones que crecía de forma cuadrática con el número de tareas. A continuación se comentan otras suposiciones algo más optimistas y que se encuentran también en sistemas procesadores reales.

Un modelo con coste de comunicaciones lineal

En este modelo se va a suponer que las tareas de un procesador se comunican con el resto de tareas del resto de procesadores, pero en vez de suponer que cada tarea de un procesador se comunica con cada tarea del resto de procesadores, lo que se va a suponer es que cada tarea de un procesador se comunica con el resto de procesadores y no con cada tarea dentro de cada procesador; el procesador ya se encarga de difundir esta comunicación entre las tareas. De esta manera el coste de comunicaciones será proporcional al coste por tarea y al número de procesadores, siendo un coste lineal con el número de tareas:

$$T_e = R \text{ máx}\{k_i\} + CN \tag{6.13}$$

Aunque la fórmula es diferente a la del modelo obtenido anteriormente, se puede demostrar que aplicando los mismos criterios de reparto utilizados entonces (reparto igualitario pero intentando que sea a la vez disparejo) se obtiene el tiempo de ejecución mínimo. La diferencia es que con este modelo hay un mayor *speed-up* disponible.

En una distribución equitativa el primer término de la ejecución es aproximadamente RM/N que decrece al aumentar N. Por otro lado, el término debido a las comunicaciones (CN) crece al aumentar N, por lo que llega un momento a partir del cual el tiempo deja de disminuir para hacerse más grande. Esto quiere decir que añadir más procesadores no sólo no disminuye el tiempo de ejecución sino que lo aumenta. El tiempo de ejecución a partir del cual añadir más procesadores empeora el rendimiento es un mínimo local de la expresión (6.13), por lo que es fácil calcular el número de procesadores umbral derivando la expresión anterior con respecto a N e igualándola a cero para calcular el mínimo:

$$-\frac{RM}{N^2} + C = 0$$

$$C = \frac{RM}{N^2}$$

dando finalmente que:

$$N_{umbral} = \sqrt{\frac{RM}{C}}$$

Esta raíz cuadrada que se obtiene es un desastre. Uno espera que M tareas puedan llevarse a cabo velozmente en $N = M$ procesadores, pero este modelo dice que debido al coste de las comunicaciones, el paralelismo efectivo se reduce a la raíz cuadrada de lo que se había previsto. Estas malas noticias se pueden mitigar con un factor R/C más alto por lo que la granularidad gruesa es mejor en este caso, aunque este efecto también se encuentra dentro de la raíz.

Estos resultados son todavía más pesimistas si se considera el coste de los procesadores extra en relación con su beneficio. Dado que el tiempo de ejecución ya no disminuye una vez alcanzado N_{umbral} se puede decir que, mucho antes de que N llegue a este umbral, se habrá alcanzado el punto donde la mejora obtenida al añadir un procesador no justifica su coste. Por ejemplo, una aplicación que en principio tiene unas 10.000 tareas se podría ejecutar como mucho en 100 procesadores para obtener el tiempo mínimo, pero sólo en unos 10 si además queremos que el sistema sea económicamente aprovechable.

El modelo presentado difiere del modelo original en el segundo término. En el modelo original el coste del segundo término crecía de forma cuadrática con la constante M. Las contribuciones al tiempo de ejecución variaban inversamente con N. Para N grande, el tiempo de ejecución se hacía del orden de $CM^2/2$ que no crece por mucho que se incremente N. Como ambos miembros de la ecuación decrecían con N el tiempo siempre decrece al aumentar el número de procesadores.

En el modelo propuesto ahora el segundo término crece con N y esto es por lo que aparece el umbral a partir del cual el rendimiento decae. Los dos modelos muestran que la penalización por sobrecarga existe y que se manifiesta limitando el uso efectivo del paralelismo. En un caso el paralelismo viene limitado por el número de tareas a ejecutar y en el otro viene limitado por el número de procesadores efectivos que son interesantes utilizar.

Un modelo optimista: comunicaciones completamente solapadas

Hasta ahora se ha supuesto que el procesador se encontraba ocupado, o bien realizando un trabajo útil R, o bien, comunicándose con el resto de procesadores y tareas. Esta suposición es cierta puesto que lo normal es que haya recursos compartidos y no se puedan hacer las dos cosas al mismo tiempo. Sin embargo, hay sistemas donde se puede considerar que mientras se está realizando la comunicación también se está llevando a cabo un trabajo útil. A continuación se propone un modelo para el caso extremo en el cual todo el coste de comunicaciones se puede llevar a cabo en paralelo con el trabajo útil.

Para este nuevo modelo se supone que si el coste debido a las comunicaciones está por debajo del trabajo útil, entonces sólo se considera el trabajo útil y no hay por tanto ningún coste adicional por el hecho de comunicar unas tareas con otras. Esto se expresa en la siguiente ecuación:

$$T_e = \text{máx}\left\{ R\,\text{máx}\left\{k_i\right\}, \frac{C}{2}\sum_{i=1}^{N} k_i(M - k_i) \right\} \tag{6.14}$$

Las gráficas de la figura 6.8 pueden servir para ver el resultado de esta ecuación para dos procesadores. En esta figura aparecen las dos componentes, una debida a las comunicaciones (parábola invertida), y la otra debida al trabajo útil (líneas rectas), el máximo formado por ambos términos da el tiempo de ejecución para este modelo optimista. Las intersecciones de ambas curvas dan las situaciones donde el tiempo de ejecución es mínimo. Si no hay intersecciones porque las comunicaciones se solapan completamente con el trabajo útil entonces el mínimo se encuentra en el reparto equitativo.

Los puntos de intersección de ambos términos se dan cuando:

$$R(M-k) = C(M-k)k$$

obteniéndose entonces el reparto:

$$k = \frac{R}{C}$$

siempre que $1 \leq k \leq M/2$.

Si se sustituye esta condición en la ecuación (6.14), el tiempo de ejecución será:

$$R(M - R/C)$$

y el *speed-up* queda como:

$$S = \frac{1}{\left(1 - \frac{R}{CM}\right)}$$

Como k está restringido en un rango, lo mismo le ocurrirá a R/C quedando $1 \leq R/C \leq M/2$. Para R/C dentro de este rango, el *speed-up* para dos procesadores está en el rango de 1 a 2 y es máximo cuando $R/C = M/2$ que es el mismo valor obtenido para el primer modelo de todos. Si no hay solapamiento completo entonces la distribución buena ya no es la igualitaria, aunque en realidad ésta se puede obtener haciendo R/C lo suficientemente grande.

Para N procesadores este modelo es fácil de analizar debido a los resultados siguientes. Para cualquier valor k_i máximo obtenido del tiempo de ejecución (término R), el reparto equitativo da el máximo tiempo de comunicaciones. Por lo tanto, la condición a partir de la cual se da el tiempo mínimo (reparto igualitario) será cuando coincidan el tiempo mínimo de ejecución y el máximo de comunicaciones:

$$\frac{RM}{N} = \frac{CM^2}{2}\left(1 - \frac{1}{N}\right)$$

que para N grande ocurre más o menos cuando:

$$\frac{R}{C} = \frac{M}{2}N$$

En este caso, para un tiempo total mínimo, el número de procesadores en función de R/C y M viene dado por la siguiente función:

$$N = \frac{2}{M}\frac{R}{C}$$

obteniéndose que la opción óptima para el número de procesadores es inversamente proporcional al número de tareas disponibles.

Si aumenta el paralelismo disponible (M) la mejor estrategia consiste en disminuir el número de procesadores. El decrecimiento de N con M viene del hecho de que el coste de la sobrecarga crece M veces más rápido que el tiempo de ejecución.

Un modelo con varios enlaces de comunicaciones

Una suposición común al resto de modelos expuestos hasta ahora, era que el paralelismo permite que el tiempo de ejecución (R) se solape entre los procesadores, pero las operaciones de sobrecarga (C) se realizaban de forma secuencial. Si se considera que las operaciones de sobrecarga son solamente las debidas a las comunicaciones, entonces estos modelos sirven para sistemas en los cuales sólo existe un canal de comunicación común para todos los procesadores. Este es el caso en el que todos los procesadores están conectados a un bus común o red en anillo o comparten una memoria común a la que se accede de forma exclusiva.

Es posible replicar los enlaces de comunicación (redes complejas) y otras características arquitectónicas que contribuyan al término de sobrecarga del modelo. Haciendo esto se obtiene que C ya no es constante sino que se convierte en una función de N.

Supongamos que se tiene un sistema en el cual los enlaces de intercomunicación crecen con N de manera que cada procesador tiene un enlace dedicado a cada uno del resto de procesadores. Con esta suposición las comunicaciones entre procesadores quedan solapadas unas con otras. Sin embargo, incluso con $\mathrm{O}(N^2)$ enlaces, todavía no es posible establecer más de $\mathrm{O}(N)$ conversaciones concurrentes porque cada procesador puede enviar o recibir información de un único procesador a un tiempo.

En este caso se puede dividir el segundo término de la ecuación (6.12) por N obteniéndose:

$$T_e = R\ \text{máx}\{k_i\} + \frac{C}{2N}\sum_{i=1}^{N} k_i(M - k_i) \tag{6.15}$$

Esta ecuación supone que un procesador está o calculando o comunicando o sin hacer nada, y que el coste total debido a las comunicaciones decrece inversamente con N, ya que pueden haber N conversaciones a un tiempo. El tiempo sin hacer nada viene en parte por el tiempo que tienen que esperar los procesadores que acaban pronto a los que les cuesta más.

Los dos términos de la ecuación (6.15) tienden a decrecer con N. Esta expresión es muy similar a la del modelo inicial de la ecuación (6.12) salvo por la aparición de N en el segundo término. Una distribución igualitaria minimiza el primer término, pero no el segundo que sigue siendo mínimo para el caso más dispar posible. Suponiendo como siempre que el reparto es igualitario, el mínimo tiempo posible será:

$$T_e = \frac{RM}{N} + \frac{CM^2}{2N}\left(1 - \frac{1}{N}\right)$$

El paralelismo es útil en este caso pero sólo hasta que el tiempo de ejecución deja de decrecer cuando se añaden nuevos procesadores. Esto quiere decir que este tiempo de ejecución alcanza un mínimo. Para calcular este mínimo se deriva T_e con respecto a N e igualamos a cero:

$$-\frac{RM}{N^2} - \frac{CM^2}{2N^2} + \frac{2CM^2}{2N^3} = 0$$

$$\frac{CM}{N} = R + \frac{CM}{2}$$

obteniéndose que:

$$N = \frac{CM}{R + \frac{CM}{2}} = \frac{2}{\frac{2R}{CM} + 1} < 2$$

Esto quiere decir que el tiempo de ejecución siempre mejora con la adición de procesadores, salvo que se tenga un procesador solamente.

Para saber si N procesadores dan mejor tiempo que un único procesador hay que igualar los tiempos de un procesador con los de varios:

$$RM = \frac{RM}{N} + \frac{CM^2}{2N}\left(1 - \frac{1}{N}\right)$$

Simplificando se obtiene que el punto a partir del cual es menor el tiempo con N procesadores se da cuando se cumple:

$$\frac{R}{C} = \frac{M}{2N}$$

En este caso el factor de granularidad R/C y N están inversamente relacionados en el umbral. Por lo tanto, cuanto más grande sea N menor granularidad se puede permitir. En el umbral la máquina paralela tiene un coste realmente alto, por un lado no se gana nada en tiempo y, por otro, el número de procesadores es del orden de N y el número de enlaces es del orden de N^2.

La conclusión de este modelo es que añadiendo enlaces al sistema (aumentando el ancho de banda de las comunicaciones) se puede permitir una granularidad menor que en otros casos. Sin embargo, esta menor granularidad genera un coste que crece más rápido que sólo el incremento del coste de procesamiento. La decisión de si el aumento de velocidad obtenido por el aumento del ancho de banda vale la pena o no, depende fuertemente de la tecnología utilizada para las comunicaciones entre procesadores.

Resumen de los modelos presentados

A continuación se resumen los hallazgos encontrados a partir de los modelos presentados:

1. Las arquitecturas multiprocesador producen un coste por sobrecarga adicional que no está presente en los mono-procesadores, procesadores vectoriales, u otros tipos de procesadores donde hay un único flujo de instrucciones. El coste por sobrecarga incluye el coste de preparación de tareas, contención en los recursos compartidos, sincronización, y comunicaciones entre procesadores.
2. Aunque el tiempo de ejecución para un trozo de programa tiende a disminuir con el número de procesadores que trabajan en ese trozo de programa. El coste por sobrecarga tiende a crecer con el número de procesadores. De hecho, es posible que el coste de la sobrecarga crezca más rápido que lineal en el número de procesadores.
3. La relación R/C es una medida de la cantidad de ejecución de programa (tiempo de ejecución útil) por unidad de sobrecarga (tiempo de comunicaciones), dentro de la implementación de un programa en una arquitectura específica. Cuando más grande sea esta relación más eficiente será la computación, ya que una porción pequeña del tiempo está dedicada a la sobrecarga. Sin embargo, si la relación R/C se hace grande al particionar el cálculo en pocos trozos grandes en vez de muchos trozos pequeños, el paralelismo disponible se reduce enormemente, lo que limita la mejora que se puede obtener de un multiprocesador.

Con esto aparece un dilema claro: por un lado R/C debe ser pequeño para poder tener un gran número de tareas potencialmente paralelas, y por otro lado, R/C debe

ser grande para evitar los costes de sobrecarga. Debido a esto no se puede esperar tener un sistema de alto rendimiento sin más que construir el multiprocesador con el mayor número de procesadores posible permitido por la tecnología.

Existe algún número máximo de procesadores por debajo del cual es coste está justificado, y este número máximo depende mayormente de la arquitectura del sistema, la tecnología utilizada (especialmente comunicaciones), y de las características de la aplicación específica que se tenga que ejecutar.