Capítulo 1

Introducción al procesamiento paralelo

En este capítulo se introducen los conceptos básicos del procesamiento paralelo en computadores de altas prestaciones. Examinaremos las diferentes formas de actividades concurrentes que aparecen en los sistemas informáticos modernos, algunas de las clasificaciones de arquitecturas paralelas propuestas así como las métricas y modelos que permiten obtener el rendimiento en este tipo de arquitecturas.

El material presentado en este capítulo introductorio proporcionará una visión global de la materia y allanará el camino al estudio detallado de las arquitecturas paralelas.

1.1 Introducción

Un computador paralelo *es una colección de elementos de procesamiento que cooperan y se comunican para solucionar rápidamente un problema grande.*

Esta simple definición da lugar, sin embargo, a muchas preguntas sobre ¿cómo de grande es dicha colección?, ¿qúe potencia tienen los elementos de proceso y si el número puede crecer de una manera sencilla?, ¿cuánta memoria tiene?, ¿cómo cooperan los elementos y se comunican?, ¿cómo se transmiten los datos entre los procesadores?, ¿qué clase de interconexión se proporciona entre los mismos y qué operaciones existen para secuenciar las acciones que se llevan a cabo en diferentes procesadores?, ¿cuáles son las primitivas de abstracción que el hardware proporciona al programador?, ¿cómo escala? y por último ¿cómo afectan todas estas cuestiones al rendimiento?

Como respuesta a estas preguntas se han propuesto numerosas estructuras de organización, a menudo ligadas a modelos de programación particulares, como forma de obtener el mejor rendimiento para una tecnología dada. El estudio de dichas estructuras, es el objetivo principal de esta asignatura.

1.2 Fuentes del paralelismo

El procesamiento paralelo tiene como principal objetivo explotar el paralelismo inherente a las aplicaciones informáticas. Todas las aplicaciones no presentan el mismo perfil cara al paralelismo: unas contienen mucho, otras muy poco. Al lado de este factor cuantitativo evidente, es necesario considerar también un factor cualitativo: la manera a través de la cual se explota el paralelismo. Cada técnica de explotación del paralelismo se denomina fuente. Distinguiremos tres fuentes principales:

- El paralelismo de control
- El paralelismo de datos
- El paralelismo de flujo

1.2.1 El paralelismo de control o funcional

La explotación del paralelismo de control proviene de la constatación natural de que en una aplicación existen acciones que podemos "hacer al mismo tiempo". Las acciones, llamadas también tareas o procesos pueden ejecutarse de manera más o menos independiente sobre unos recursos de cálculo llamado también procesadores elementales (o PE). La figura 1.1 muestra el concepto que subyace tras el paralelismo de control

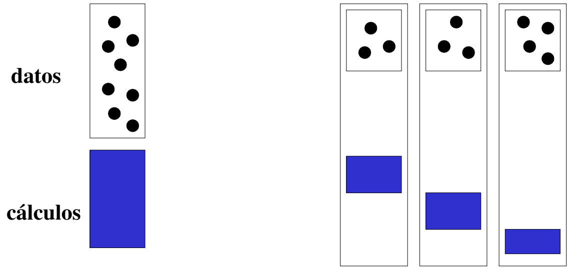


Figura 1.1: Paralelismo de control.

En el caso en que todas las acciones sean independientes es suficiente asociar un recurso de cálculo a cada una de ellas para obtener una ganancia en tiempo de ejecución que será lineal: N acciones independientes se ejecutarán N veces más deprisa sobre N PE que sobre uno solo. Este es el caso ideal, pues las acciones de un programa real suelen presentar dependencias entre ellas. Distinguiremos dos clases de dependencias:

- Dependencia de control de secuencia: corresponde a la secuenciación en un algoritmo clásico.
- Dependencia de control de comunicación: una acción envía informaciones a otra acción.

La explotación del paralelismo de control consiste en administrar las dependencias entre las acciones de un programa para obtener así una asignación de recursos de cálculo lo más eficaz posible. La figura 1.2 muestra un ejemplo de paralelismo de control aplicado a la ejecución simultánea de instrucciones.

1.2.2 El paralelismo de datos

La explotación del paralelismo de datos proviene de la constatación natural de que ciertas aplicaciones trabajan con estructuras de datos muy regulares (vectores, matrices) repitiendo una misma acción sobre cada elemento de la estructura. Los recursos de cálculo se asocian entonces a los datos. A

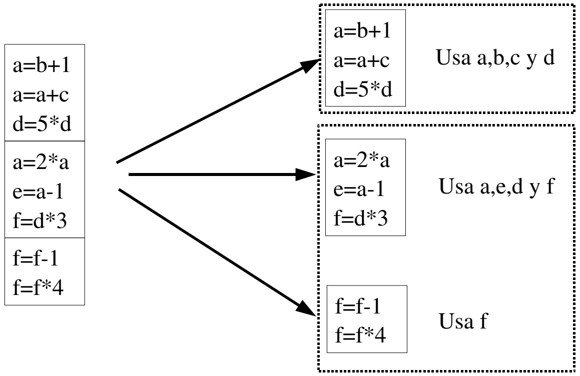


Figura 1.2: Ejemplo de paralelismo de control.

menudo, existe un gran número de datos idénticos: millares o hasta millones. Si el número de PE es inferior al de datos, éstos se repartirán sobre los PE disponibles. La figura 1.3 muestra de forma gráfica el concepto de paralelismo de datos.

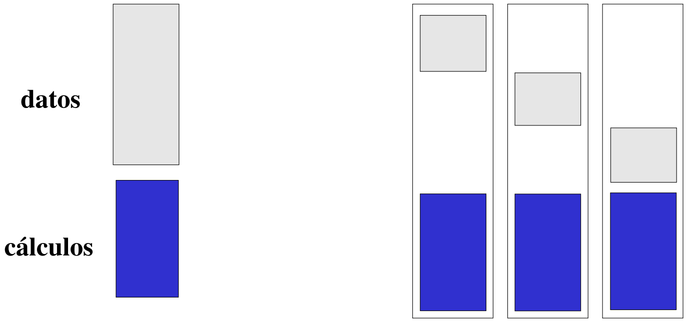


Figura 1.3: Parelelismo de datos.

Como las acciones efectuadas en paralelo sobre los PE son idénticas, es posible centralizar el control. Siendo los datos similares, la acción a repetir tomará el mismo tiempo sobre todos los PE y el controlador podrá enviar, de manera síncrona, la acción a ejecutar a todos los PE.

Las limitaciones de este tipo de paralelismo vienen dadas por la necesidad de dividir los datos vectoriales para adecuarlos al tamaño soportado por la máquina, la existencia de datos escalares que limitan el rendimiento y las existencia de operaciones de difusión (un escalar se reproduce varias veces convirtiéndose en un vector) que no son puramente paralelas. En la figura 1.4se muestra un ejemplo de paralelismo de datos.

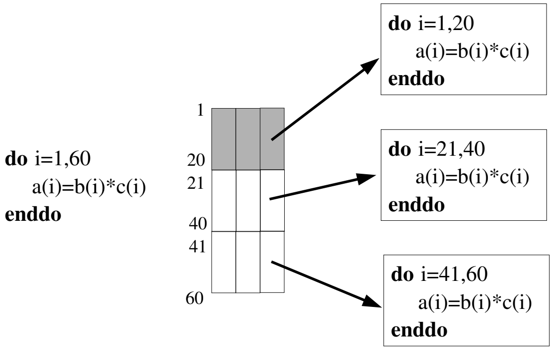


Figura 1.4: Ejemplo de la aplicación del paralelismo de datos a un bucle.

1.2.3 El paralelismo de flujo o pipelining

La explotación del paralelismo de flujo proviene de la constatación natural de que ciertas aplicaciones funcionan en modo cadena: disponemos de un flujo de datos, generalmente semejantes, sobre los que debemos efectuar una sucesión de operaciones en cascada. La figura 1.5 muestra de forma gráfica el concepto de paralelismo de datos.

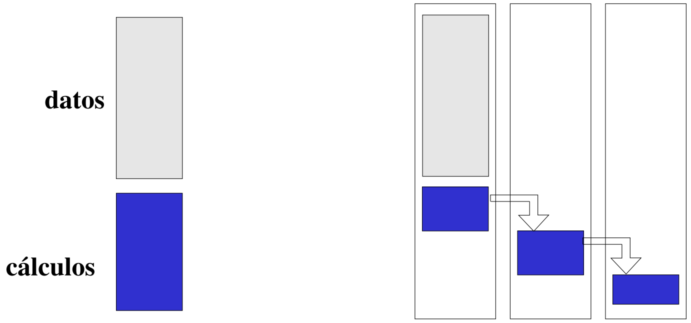


Figura 1.5: Paralelismo de flujo.

Los recursos de cálculo se asocian a las acciones y en cadena, de manera que los resultados de las acciones efectuadas en el instante t pasen en el instante $t+1$ al PE siguiente. Este modo de funcionamiento se llama también segmentación o pipeline.

El flujo de datos puede provenir de dos fuentes:

- Datos de tipo vectorial ubicados en memoria. Existe entonces una dualidad muy fuerte con el caso del paralelismo de datos.

- Datos de tipo escalar provenientes de un dispositivo de entrada. Este dispositivo se asocia a menudo a otro de captura de datos, colocado en un entorno de tiempo real.

En ambos casos, es igual la ganancia obtenida en relación al número de etapas (número de PE). Todos los PE no estarán ocupados mientras el primer dato no haya recorrido todo el cauce, lo mismo ocurrirá al final del flujo. Si el flujo presenta frecuentes discontinuidades, las fases transitorias del principio y del fin pueden degradar seriamente la ganancia. La existencia de bifurcaciones también limita la ganancia obtenida.

1.3 Clasificación de las arquitecturas avanzadas

Existen diversos esquemas de clasificación de los sistemas de computación basados en su estructura o comportamiento. Los principales métodos de clasificación consideran el número de conjuntos de instrucciones y/o operadores que pueden se procesados simultáneamente, la organización interna de los procesadores, la estructura de la conexión interprocesador, o los métodos empleados para calcular los flujos de instrucciones y datos dentro del sistema.

1.3.1 Clasificación de Flynn

En 1966, M. J. Flynn propuso una clasificación de los ordenadores en función del análisis de los flujos de control (*instruction stream*) y de datos (*data stream*). Estos flujos pueden ser simples o múltiples, definiendo así cuatro clases de máquinas como se muestra en la figura 1.6.

a) La arquitectura SISD de Flynn.

A esta categoría pertenece el modelo de máquina secuencial propuesto por von Neumann. Las instrucciones son ejecutadas secuencialmente, pero esta ejecución puede estar segmentada, es decir, puede haber superposición de varias instrucciones encontrándose en diferentes etapas de ejecución. La figura 1.6a muestra este esquema.

b) La arquitectura SIMD de Flynn.

Esta categoría incluye las máquinas tales como el Illiac IV, que tiene una única unidad de control del programa pero múltiples unidades de ejecución independientes (procesador matricial). Todas las unidades de procesamiento reciben la misma instrucción radiada por la unidad de control, pero esta instrucción se ejecuta sobre diferentes conjuntos de datos. La figura 1.6b muestra este esquema.

c) La arquitectura MISD de Flynn.

Existe una colección de unidades de procesamiento que ejecutan diferentes instrucciones pero sobre el mismo flujo de datos. Es decir, los resultados de un procesador se convierten en los operandos del siguiente. Algunos investigadores consideran impracticable esta categoría, mientras otros incluyen en ella a los computadores vectoriales y los sistólicos. La figura 1.6c muestra este esquema.

d) La arquitectura MIMD de Flynn.

Esta categoría cubre a los multiprocesadores y a los multicomputadores, es decir, sistemas con más de un procesador capaz de ejecutar varios programas simultáneamente. La figura 1.6d muestra este esquema.

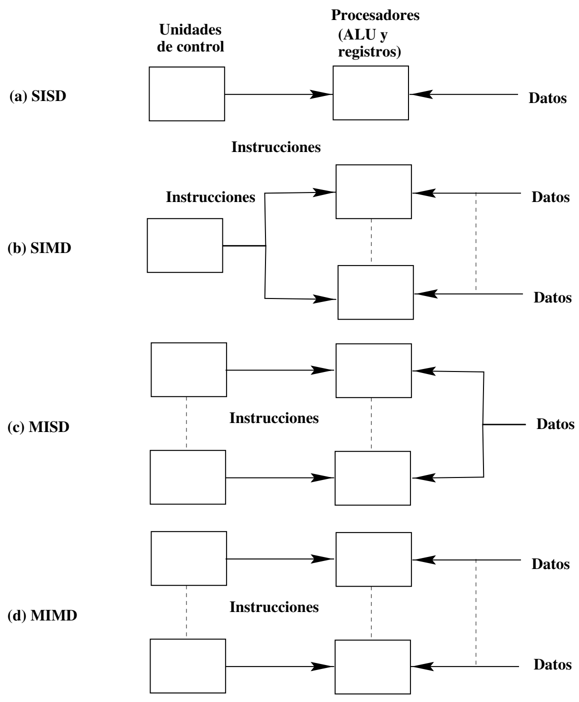


Figura 1.6: La clasificación de Flynn.

1.3.2 Modelos de computadores paralelos

La clasificación de Flynn se ha mostrado como un buen método para clasificar las arquitecturas de ordenadores durante tres décadas. Sin embargo, los avances en las tecnologías han dado lugar a arquitecturas que no encajan de forma clara dentro de la taxonomía de Flynn. Por ejemplo, los procesadores vectoriales (SIMD y MISD) y las arquitecturas híbridas. Para solucionar este problema, varios autores han propuesto taxonomías alternativas.

Una de esas taxonomías, para computadores paralelos, es la que se muestra a continuación y está basada en modelos de computadores paralelos

Un *modelo de máquina paralela* (también conocido como modelo de programación, arquitectura tipo, modelo conceptual o modelo idealizado) es un computador paralelo abstracto desde el punto de vista del programador, análogo al modelo de von Neumann para computadores secuenciales.

1.3.2.1 Modelos abstractos de máquinas

Un modelo de máquina abstracta se usa principalmente en el diseño y análisis de algoritmos paralelos sin preocuparse de los detalles de la máquina física. Veremos tres modelos.

El modelo PRAM. Una PRAM (*Parallel Random-Access Machine*) de tamaño *n*, como se muestra en la figura 1.7a, está formada por *n* procesadores, todos con acceso a la memoria compartida. Cada paso de tiempo básico se denomina *ciclo*. En un ciclo, cada procesador ejecuta una instrucción. Podría ser una instrucción de no hacer nada, en cuyo caso se dice que el procesador está ocioso en ese ciclo.

Todos los procesadores implícitamente se sincronizan en cada ciclo, y este gasto en sincronización se supone que es cero. La comunicación se realiza a través de la lectura y escritura en variables compartidas. El gasto de comunicación también se ignora. Así, el único overhead (gasto) que se tiene en cuenta en un programa PRAM es el desbalanceo de la carga.

Debido a su simplicidad, este modelo se ha usado ampliamente en muchos de los algoritmos paralelos teóricos, así como en el analisis de la complejidad de los algoritmos paralelos

El modelo BSP. El BSP (*Bulk Synchronous Parallel*) está formado por *n* pares procesador/memoria (nodos) que están interconectados mediante una red como se muestra en la figura 1.7b. Un programa BSP tiene *n* procesos, cada uno residiendo en un nodo. La unidad de tiempo básico es el ciclo. El programa se ejecuta según una secuencia fija de superpasos. En cada superpaso, un proceso ejecuta los cálculos en como mucho *w* ciclos, las operaciones de comunicación llevan *gh* ciclos, y una barrera de sincronización que necesita *l* ciclos.

El computador BSP es un sistema MIMD, ya que los procesos pueden ejecutar distintas instrucciones al mismo tiempo. No requiere ningún mecanismo de interacción específico, puede usar o variables compartidas o paso de mensajes. Hay un espacio de direcciones simple, y cada procesador puede acceder tanto a su memoria local como a la memoria remota de cualquier otro nodo.

El modelo de fases paralelas. Intenta refinar los modelos anteriores, siendo muy similar al modelo BSP con las siguientes distinciones:

- Un programa paralelo se ejecuta como una secuencia de fases. La siguiente fase no puede empezar hasta que todas las operaciones de la fase actual hayan acabado. Hay tres tipos de fases:

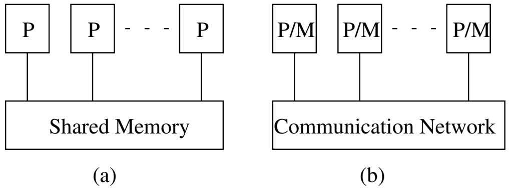


Figura 1.7: Modelos teóricos de computadores paralelos. a)PRAM y b)BSP

1. *Fase paralela*: Se refiere al gasto de trabajo involucrado en el manejo de procesos, tales como creación de procesos.
2. *Fase de cálculo*: Uno o más procesadores ejecutan un número de operaciones locales. Aquí local significa que todos los datos necesarios por un nodo están disponibles en su memoria local.
3. *Fase de interacción*: Se refiere a las tareas necesarias para realizar la oepración de interacción: una comunicación, una sincronización, etc.

- Para una determinada fase de cálculo, denotamos la carga de trabajo (en Mflop) por w y t_f es el tiempo medio (en μs) para ejecutar un cálculo. Diferentes fases de cálculo pueden ejecutar diferentes cargas de trabajo a diferentes velocidades, correspondiendo a diferentes valores de w y t_f. Aquí está la diferencia con los modelos anteriores, donde cada operación de cálculo se asumía igual a un ciclo.
- Diferentes operaciones de interacción pueden llevar diferentes tiempos.

En la figura 1.8 se muestra un esquema sobre el funcionamiento de este modelo.

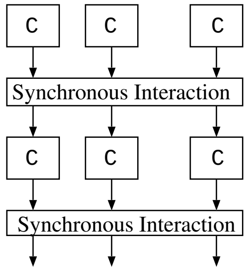


Figura 1.8: Modelo de fases paralelas (C:cálculo).

1.3.2.2 Modelos físicos de máquinas

Los sistemas de computadores de gran escala se clasifican, generalmente, en seis modelos prácticos: Las máquinas SIMD (*Single-Instruction Multiple-Data*), PVP (*Parallel Vector Processor*), SMP

(*Symmetric MultiProcessor*), MPP (*Massively Parallel Processor*), COW (*Cluster Of Workstations*) y DSM (*Distributed Shared Memory*). Los modelos SIMD se suelen utilizar para aplicaciones de propósito específico, mientras que el resto son todos modelos MIMD, como se resume en la tabla 1.1.

Attributes	PRAM	PVP/SMP	DSM	MPP/COW
Homogeneity	MIMD	MIMD	MIMD	MIMD
Synchrony	Instruction-level synchronous	Asynchronous or loosely synchronous	Asynchronous or loosely synchronous	Asynchronous or loosely synchronous
Interaction mechanism	Shared variable	Shared variable	Shared variable	Message passing
Address space	Single	Single	Single	Multiple
Access cost	UMA	UMA	NUMA	NORMA
Memory model	EREW, CREW or CRCW	Sequential consistency	Weak ordering is widely used	N/A
Example machines	Theoretical model	IBM R50, Cray T-90	Stanford DASH, SGI Origin 2000	Cray T3E, Berkeley NOW

Tabla 1.1: Atributos de los modelos de máquinas paralelas.

En la figura 1.9 se muestran los cinco modelos de máquinas paralelas MIMD.

Procesadores Paralelos Vectoriales (PVP): La estructura de un PVP típico se muestra en la figura 1.9a. Ejemplos de PVPs son el Cray C-90, Cray T-90 y NEC SX-4. Estos sistemas contienen un número pequeño de procesadores vectoriales potentes, cada uno de como mínimo 1Gflos de rendimiento. Los procesadores se conectan a un determinado número de módulos de memoria compartida, que proporcionan acceso a los datos a gran velocidad. Estas máquinas, normalmente no usan caches, y en su lugar usan un gran número de registros vectoriales y buffers de instrucción.

Multiprocesadores simétricos (SMP): Su estructura se muestra en la figura 1.9b. Ejemplos de SMPs son el IBM R50, el Challenge Power SGI, y el Alpha DEC 8400. Suelen usan microprocesadores con caches internas y externas. Estos procesadores se conectan a una memoria compartida a través de un bus de alta velocidad. La mayoría de sistemas tienen hasta 64 procesadores como mucho, ya que el uso de una memoria compartida centralizada y un bus impiden escalar mucho el sistema.

Procesadores masivamente paralelos (MPP): Normalmente hace referencia a un computador de gran escala con las siguientes características:

- Usa microprocesadores en los nodos de procesamiento.
- Usa memoria físicamente distribuida sobre los nodos de procesamiento.
- Usa una red de interconexión con gran ancho de banda para comunicación y baja latencia.
- Puede escalar hasta cientos o incluso miles de procesadores.
- Es una máquina MIMD asíncrona. Sin embargo, los procesos se sincronizan por medio de operaciones de paso de mensajes.
- El programa consiste de múltiples procesos, en el que cada uno tiene su espacio de direcciones privado.

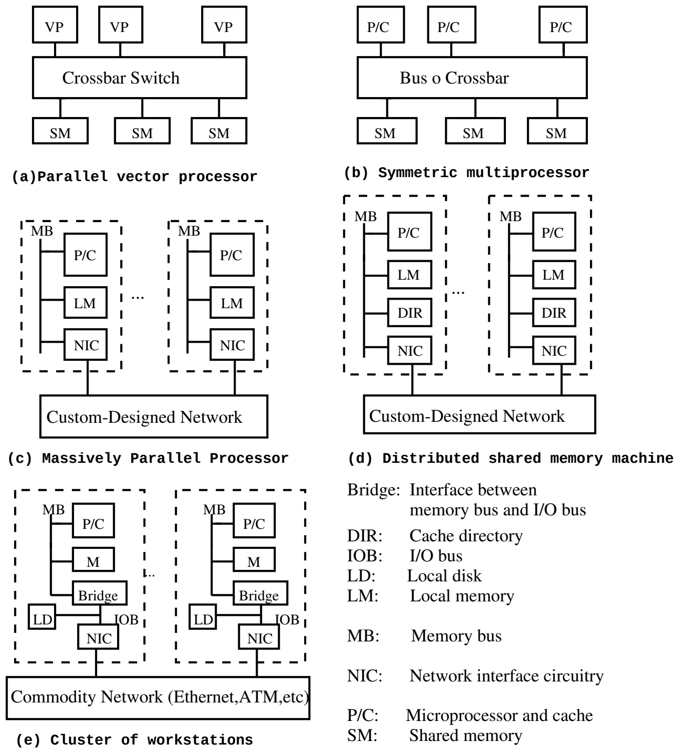


Figura 1.9: Modelos físicos de computadores paralelos.

El MPP modelado en la figura 1.9c es más restrictivo, representando máquinas como el Intel Paragon, que están formadas por un número de nodos de procesamiento, cada uno conteniendo uno o más microprocesadores .

Máquinas de memoria compartida distribuida (DSM): Las máquinas DSM se modelan en la figura 1.9d, basada en la arquitectra DASH de Stanford. El directorio cache (DIR) se usa para soportar caches de coherencia distribuida. La diferencia principal con los SMP es que la memoria está físicamente distribuida entre los diferentes nodos. Sin embargo, el sistema software y hardware crean la ilusión de un sólo espacio de direcciones para las aplicaciones de los usuarios.

Clusters de computadores (COW): El concepto de cluster se muestra en la figura 1.9e. Ejemplos incluyen el IBM SP2, y la NOW de Berkeley. Los cluster son la variante de bajo coste de los MPPs. Las distinciones más importantes son:

- Cada nodo de una COW es una estación de trabajo completa menos algunos periféricos. Un nodo puede ser un SMP o un PC.
- Los nodos están conectados a través de una red de bajo coste, como Ethernet, FDDI, conmutador ATM.
- La interfase de la red es débilmente acoplada al bus de E/S en un nodo.
- Hay siempre un disco local, que puede estar ausente en un nodo MPP.
- Un sistema operativo completo reside en cada nodo, mientras que en algunos MPPs sólo existe un microkernel.

Por último, en la figura 1.10 se muestra una última clasificación en relación con el modelo de memoria que soporta la máquina.

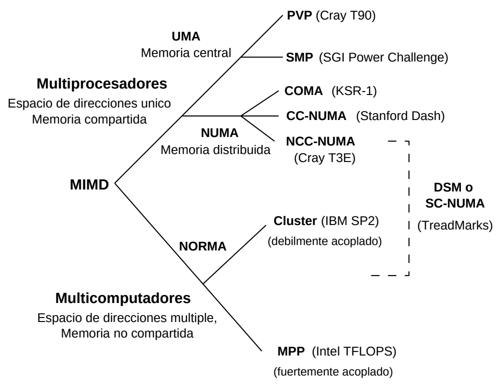


Figura 1.10: Diferentes tipos de arquitecturas de memoria para construir computadores paralelos

1.4 Medidas de rendimiento

En esta sección se definirán algunas de las medidas más utilizadas a la hora de determinar el rendimiento de una arquitectura paralela. Así, introduciremos los conceptos de: speedup, eficiencia de un sistema, utilización, redundancia, etc.

1.4.1 Perfil del paralelismo en programas

El grado de paralelismo refleja como 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 speedup ideal en una máquina con recursos ilimitados.

Grado de paralelismo. 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 usados para ejecutar el programa se define como el *grado de paralelismo* (GDP).

La gráfica 1.11 que muestra el GDP en función del tiempo se 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 1.11 se muestra un ejemplo de perfil del paralelismo del algoritmo divide y vencerás.

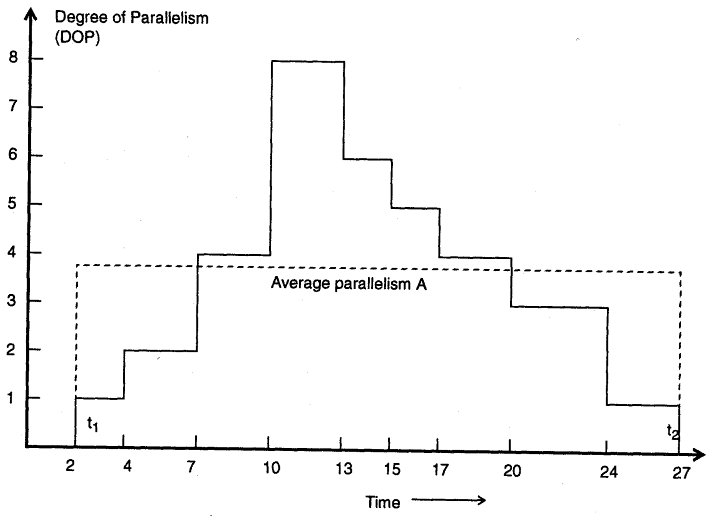


Figura 1.11: Perfil del paralelismo de un algoritmo divide y vencerás.

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

Paralelismo medio. Consideremos un procesador paralelo compuesto por n elementos de proceso homogéneos. El paralelismo máximo en un perfil es m. En el caso ideal, $n \gg m$. La *capacidad de cómputo* Δ de un solo procesador viene dada por su velocidad de ejecución (MIPS o Mflops) sin considerar las penalizaciones debidas al acceso a memoria, latencia de las comunicaciones, o sobrecarga del sistema. Cuando i procesadores estén ocupados durante un periodo de tiempo, tendremos que $GDP = i$.

La cantidad de trabajo realizado, W, es proporcional al área bajo la curva del perfil

$$W = \Delta \int_{t_1}^{\hat{t}_2} GDP(t)dt$$

Esta integral se puede calcular 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*, A, será por tanto

$$A = \frac{1}{t_2 - t_1} \int_{t_1}^{\hat{t}_2} GDP(t)dt$$

o en su forma discreta

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

Speedup asintótico. Si denotamos por $W_i = i\Delta t_i$ al trabajo realizado con $GDP = i$, entonces podemos escribir $W = \sum_{i=1}^{m} W_i$. 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* 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 *speedup asintótico* S_∞ se define como el cociente de $T(1)$ entre $T(\infty)$:

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

Si comparamos esta fórmula con la del paralelismo medio, A, observamos que $S_\infty = A$ en el caso ideal. En general, $S_\infty \leq A$ si se consideran la latencia de comunicaciones y otras sobrecargas del sistema. Obsevar que tanto S_∞ como A están definidos bajo la suposición de que $n = \infty$ o $n \gg m$.

1.4.2 Rendimiento armónico medio

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

Los diferentes modos de ejecución pueden corresponderse con procesamientos escalares, vectoriales, secuenciales o paralelos con diferentes partes del programa. Cada programa puede ejecutarse mediante una combinación de estos modos. El rendimiento *armónico medio* 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. Sean $\{R_i\}$ las velocidades de ejecución de los programas $i = 1, 2, \ldots, m$. La *media aritmética de la velocidad de ejecución 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 $\pi = \{f_i : i = 1, 2, \ldots, m\}$, definimos la *media aritmética con pesos de la velocidad de ejecución* 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 benchmark.

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

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

Con una distribución de pesos $\pi = \{f_i : i = 1, 2, \ldots, m\}$, podemos definir una *media geométrica con pesos de la velocidad de ejecución* como:

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

La media geométrica tampoco capta el rendimiento real, dado que no presenta una relación inversa con el tiempo total.

Rendimiento armónico medio. 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 benchmark se define por $R_h = \frac{1}{T_a}$:

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

Además, el rendimieno medio armónico está relacionado con el tiempo medio de ejecución. Si consideremos una distribución de pesos, podemos definir el *rendimiento armónico medio con pesos* como:

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

Speedup 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 uniprocesador con una velocidad (ratio) de ejecución $R_1 = 1$. Entonces $T_i = 1/R_i = 1/i$ es el tiempo de ejecución usando *i* procesadores con un ratio de ejecución combinado 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 $w = \{f_i : i = 1, 2, \ldots, n\}$. El *speedup armónico medio con pesos* se define como:

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

donde $T^* = 1/R_h^*$ es la *media aritmética con pesos del tiempo de ejecución* para *n* modos de ejecución.

La figura 1.12 muestra el comportamiento del speedup para tres funciones con distintos pesos.

1.4.3 Eficiencia, utilización y calidad

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

Eficiencia del sistema. Sea $O(n)$ el número total de operaciones elementales realizadas por un sistema multiprocesador de *n* elementos de proceso y $T(n)$ el tiempo de ejecución. En general, $T(n) < O(n)$ si los *n* procesadores realizan más de una operación por unidad de tiempo, donde

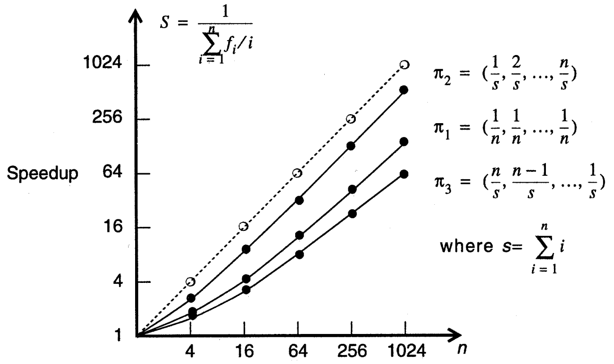


Figura 1.12: Media armónica del speedup con respecto a tres distribuciones de probabilidad: π_1 para la distribución uniforme, π_2 en favor de usar más procesadores y π_3 en favor de usar menos procesadores.

$n \geq 2$. Supongamos que $T(1) = O(1)$ en un sistema uniprocesador. El *factor speedup* 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 speedup conseguido frente al valor máximo. Dado que $1 \leq S(n) \leq n$, tenemos $1/n \leq E(n) \leq 1$.

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

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

Este ratio 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, 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 speedup y la eficiencia e inversamente relacionado con 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 speedup $S(n)$.

En resumen, usamos el speedup $S(n)$ para indicar el grado de ganancia en la velocidad en una computación paralela. La eficiencia $E(n)$ mide la porción útil del trabajo total realizado por n procesadores. La redundancia $R(n)$ mide el alcance del incremento de carga. La utilización $U(n)$ indica el alcance de la utilización de recursos que se utilizan durante un cálculo paralelo. Finalmente, la calidad $Q(n)$ combina el efecto del speedup, eficiencia y redundancia en una única expresión que indica el mérito de un cálculo paralelo sobre un sistema.

1.5 Análisis de la escalabilidad y el speedup

En esta sección describiremos tres modelos de medición del speedup. La ley de Amdahl (1967) se basa en una carga de trabajo fijo o en un problema de tamaño fijo. La ley de Gustafson (1987) se aplica a los problemas escalables, donde el tamaño del problema se incrementa al aumentar el tamaño de la máquina. El modelo de speedup de Sun y Ni (1993) es para problemas escalables limitados por la capacidad de la memoria. En la figura 1.13 se muestra un esquema de los tres modelos utilizados.

1.5.1 Ley de Amdahl para una carga de trabajo fija

En muchas aplicaciones prácticas que se centran en tiempo de respuesta, la carga de trabajo se mantiene fija con un problema de tamaño fijo. Al incrementarse el número de procesadores en un computador paralelo, la carga fija se distribuye entre más procesadores para la ejecución paralela. Por lo tanto, el objetivo principal es producir los resultados lo más pronto posible. En otras palabras, el tiempo de respuesta es nuestra principal meta. A la ganancia de tiempo obtenida para este tipo de aplicaciones se le denomina *speedup bajo carga fija*.

Considerar un problema con una carga fija *W*. Se asume que la carga *W* se puede dividir en dos partes: $W = \alpha W + (1-\alpha)W$, donde el pordentaje α de *W* se debe ejecutar secuencialmente, y el porcentaje restante, 1-α se puede ejecutar en *n* nodos simultáneamente. Si se asume que todos los overheads (penalización) se ignoran, entonces el speedup de carga fija se define como:

$$S_n = \frac{W}{\alpha W + (1-\alpha)(W/n)} = \frac{n}{1+(n-1)\alpha} \rightarrow \frac{1}{\alpha} \; cuando \; n \rightarrow \infty$$

A esta expresión se le conoce como la ley de Amdahl, y es una de las leyes fundamentales en el estudio de sistemas paralelos.

La figura 1.14 muestra el valor de speedup en función de *n* para cuatro valores de α.

La ley de Amdahl tiene varias implicaciones:

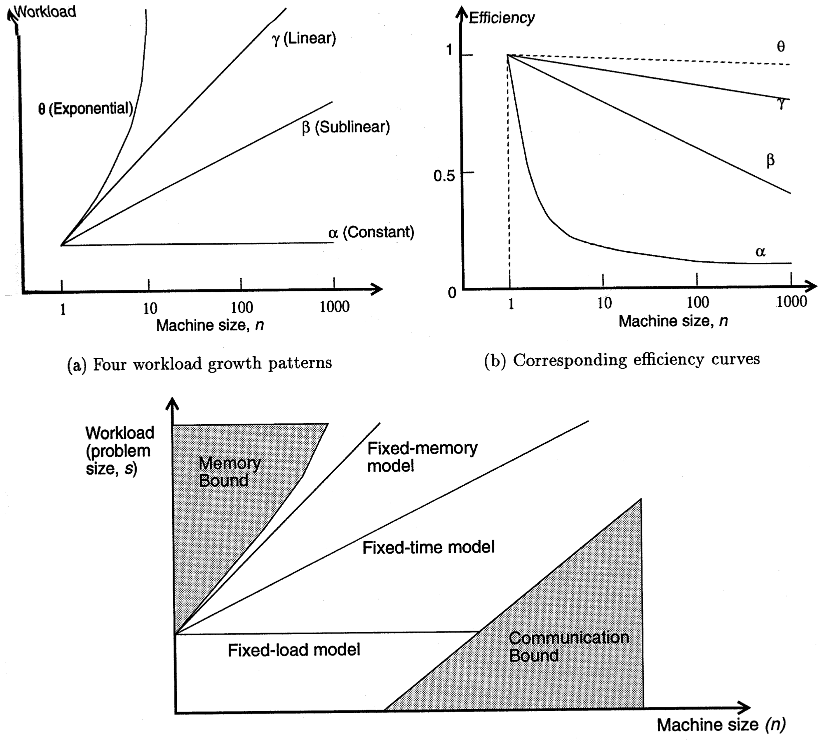


Figura 1.13: Modelos de rendimiento del speedup.

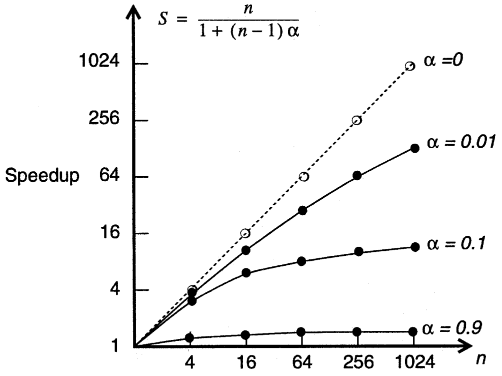


Figura 1.14: Rendimiento del Speedup para diferentes valores de α, donde α es la fracción del cuello de botella secuencial.

(1) Para una carga dada, el speedp máximo tiene un cota superior de $1/\alpha$. En otras palabras, la componente secuencial del programa es un cuello de botella. Este α se conoce como *cuello de botella secuencial de un programa*. Cuando α crece, el speedup decrece proporcionalmente.

(2) Para conseguir un buen speedup es importante hacer que el cuello de botella secuencial α sea lo más pequeño posible.

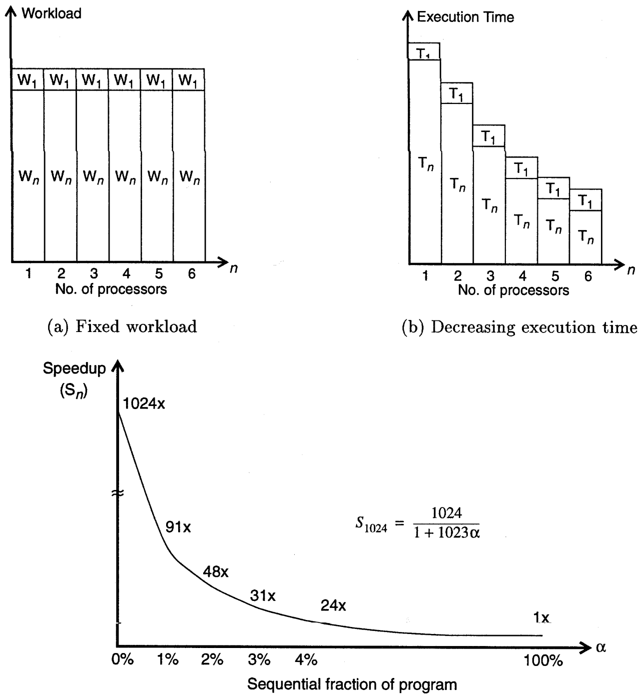
(3) Cuando un problema está formado por las dos partes, deberiamos hacer que la parte más larga se ejecutara más rápido. En otras palabras, deberiamos optimizar el caso común.

Existen una gran cantidad de factores que hemos ignorado que pueden rebajar el speedup. 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. Ahora vamos a incorporar a la ley de Amdahl todos estos overheads (T_0). Así el speedup de carga fija S_n se convierte en:

$$S_n = \frac{W}{\alpha W + (1-\alpha)(W/n) + T_0} = \frac{n}{1+(n-1)\alpha + \frac{nT_0}{W}} \to \frac{1}{\alpha + \frac{T_0}{W}} \; cuando \; n \to \infty$$

Esta extensión de la ley de Amdahl dice que no sólo debemos reducir el cuello de botella secuencial sino también incrementar la granularidad media para reducir el impacto adverso del overhead. En otras palabras, el rendimiento de un programa paralelo está limitado no sólo por el cuello de botella secuencial sino también por el overhead medio.

La figura 1.15 muestra el modelo de speedup de carga fija y la ley de Amdahl, donde $W_1 = \alpha$ y $W_n = (1-\alpha)$. La figura 1.15c muestra una gráfica de la ley de Amdahl para diferentes valores de α. El máximo speedup, $S_n = n$, se obtiene para $\alpha = 0$. El mínimo speedup, $S_n = 1$, se obtiene para $\alpha = 1$. Cuando $n \to \infty$, el valor límite es $S_n \to 1/\alpha$. La curva del speedup en la figura 1.15c cae



(a) Fixed workload

(b) Decreasing execution time

(c) Speedup with a fixed load

Figura 1.15: Modelo de speedup de carga fija y la ley de Amdahl.

rápidamente al aumentar α. Esto implica que el speedup está acotado superiormente por $1/\alpha$, cuando el tamaño de la máquina aumenta.

El speedup de carga fija cae conforme el cuello de botella secuencial α y el overhead medio crecen. El problema del cuello de botella secuencial no se puede resolver sólo con incrementar el número de procesadores del sistema. Esta propiedad ha impuesto una visión muy pesimista del procesamiento paralelo en las pasadas dos décadas.

De hecho, se observaron dos impactos en la industria de los computadores paralelos. En primer lugar, los manufacturadores 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.

1.5.2 Ley de Gustafson para problemas escalables

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 la escalabilidad en el rendimiento. Aunque el cuello de botella secuencial es un problema importante, el mismo puede aliviarse en gran medida eliminando la restricción de la carga fija (o tamaño del problema fijo). John Gustafson (1988) ha propuesto un concepto de tiempo fijo que da lugar a un modelo del speedup escalado.

Escalamiento para conseguir una mayor precisión. Las aplicaciones de tiempo crítico son la principal causa que da lugar al desarrollo de un modelo de speedup 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.

Speedup de tiempo fijo. En aplicaciones de precisión crítica, deseamos resolver el problema de mayor tamaño posible en una máquina mayor con el mismo tiempo de ejecución de resolver un problema menor en una máquina menor. Al aumentar el tamaño de la máquina, también podemos incrementar la carga de trabajo. Dado el problema original con una carga W, de la cual el porcentaje α es secuencial y el porcentaje $1-\alpha$ se puede ejecutar en paralelo.

En un sólo nodo, el tiempo de ejecución es *W*. En una máquina de *n* nodos, escalamos la carga a $W' = \alpha W + (1-\alpha)nW$. El tiempo de ejecución paralelo en *n* nodos sigue siendo *W*. Sin embargo, el tiempo secuencial para ejecutar la carga escalada es $W' = \alpha W + (1-\alpha)nW$.

El speedup de tiempo fijo para la carga escalada se define como

$$S'_n = \frac{Tiempo\ secuencial\ para\ una\ carga\ escalada}{Tiempo\ paralelo\ para\ una\ carga\ escalada}$$

$$= \frac{\alpha W + (1-\alpha)nW}{W} = \alpha + (1-\alpha)n$$

Esta ecuación se conoce como la *ley de Gustafson,* y sostiene que el speedup de tiempo fijo es una función lineal de *n*, si la carga se escala para mantener un tiempo de ejecución fijo. Cuando el problema

puede escalar hasta emparejar la potencia de cómputo disponible, la fracción secuencial no supone un cuello de botella muy grande. Hay que tener en cuenta que para que la ley de Gustafson se mantenga, es importante no sólo escalar la parte paralelizable de la carga desde $(1-\alpha)W$ a $(1-\alpha)nW$ sino que la parte secuencial αW se mantenga igual.

Speedup escalado. Ahora vamos a incorporar a la ley de Gustafson todos los overheads. Así, el speedup escalado es:

$$S'_n = \frac{\alpha W + (1-\alpha)nW}{W + T_0} = \frac{\alpha + (1-\alpha)n}{1 + T_0/W}$$

Notar que los programas paralelos con carga escalada $W' = \alpha W + (1-\alpha)nW$ tienen el mismo tiempo de computación fijo que el tiempo secuencial para la carga original *W*. El tiempo de ejecución total para el programa paralelo también incluye el overhead T_0. Este overhead es una función de *n*.

Puesto que T_0 puede tener componentes que sean crecientes, decrecientes o constantes con respecto a *n*, la ley de Gustafson puede conseguir un speedup lineal controlando el overhead como una función decreciente respecto a *n*. Esto a veces es difícil de conseguir.

La figura 1.16a muestra la relación del escalamiento de la carga de trabajo con el speedup escalado de Gustafson. Al igual que antes, $\alpha = W_1$ y $1-\alpha = W_n$.

Obsérvese que la pendiente de la curva S_n en la figura 1.16 es mucho más plana que en la figura 1.15. 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.

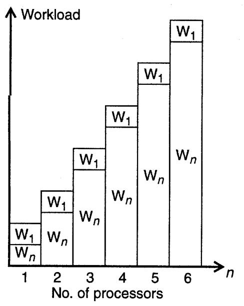
1.5.3 Modelo del speedup limitado por la memoria

Xian-He Sun y Lionel Ni (1993) han desarrollado un modelo del speedup 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 speedup, mayor precisión y mejor utilización de los recursos.

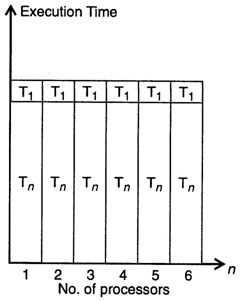
Problemas limitados por la memoria. Los cálculos científicos y las aplicaciones de ingeniería suelen necesitar una gran cantidad de memoria. De hecho, mcuhas aplicaciones de los ordenadores paralelos surgen de la limitación de la memoria más que de la CPU o la E/S. Esto es especialmente cierto en sistemas multicomputador con memoria distribuida.

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

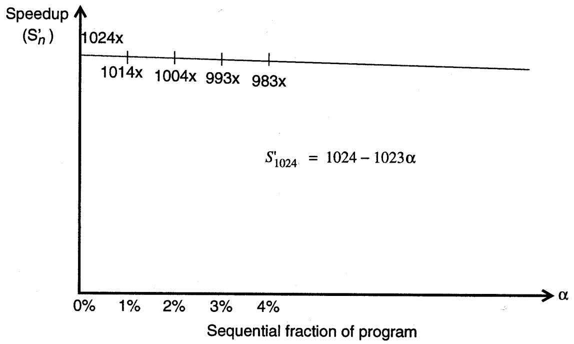
En lugar de manter fijo el tiempo de ejecución, podemos desear usar toda la memoria disponible para aumentar aún más el tamaño del problema. En otras palabras, si tienes un espacio de memoria adecuado y el problema escalado cumple el límite de tiempo impuesto por la ley de Gustafson, se puede incrementar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



(a) Scaled workload



(b) Fixed execution time



(c) Speedup with fixed execution time

Figura 1.16: Modelo de Speedup de tiempo fijo y la ley de Gustafson.

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.

Speedup de memoria fija. En un multicomputador, la capacidad total de memoria se incrementa linealmente con el número de nodos disponibles. Sea M la capacidad de memoria de un sólo nodo. En un MPP de n nodos, la capacidad total de memoria es nM.

Dado un problema limitado por la memoria, asume que utiliza toda la capacidad de memoria M de un nodo y se ejecuta en W sg (o sea, la carga secuencial es W). Como hasta ahora, la carga tiene una parte secuencial y una parte paralelizable: $W = \alpha W + (1-\alpha)W$.

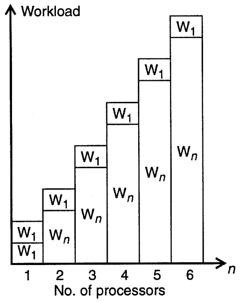
Cuando se usan n nodos, se puede resolver un problema más grande debido al incremento de memoria a nM. Vamos a suponer que la porción paralela de la carga puede ser escalada hasta $G(n)$ veces. O sea, la carga escalada es $W = \alpha W + (1-\alpha)G(n)W$. El factor $G(n)$ refleja el incremetno en la carga al aumentar la memoria n veces. El *speedup acotado por la memoria* se define como:

$$S_n^* = \frac{Tiempo\ secuencial\ para\ una\ carga\ escalada}{Tiempo\ paralelo\ para\ una\ carga\ escalada}$$

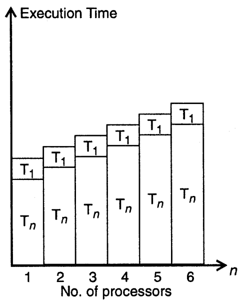
$$= \frac{\alpha W + (1-\alpha)G(n)W}{\alpha W + (1-\alpha)G(n)W/n} = \frac{\alpha + (1-\alpha)G(n)}{\alpha + (1-\alpha)G(n)/n}$$

El Tiempo secuencial se obtiene asumiendo un nodo imaginario con una capacidad de memoria de nM. Cuando se consideran todos los overheads, el speedup es:

$$S_n^* = \frac{\alpha W + (1-\alpha)G(n)W}{\alpha W + (1-\alpha)G(n)W/n + T_0} = \frac{\alpha + (1-\alpha)G(n)}{\alpha + (1-\alpha)G(n)/n + T_0/W}$$



(a) Scaled workload



(b) Slightly increased execution time

Figura 1.17: Modelo de speedup de memoria fija.

Existen tres casos especiales en donde es de aplicación la ecuación anterior:

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

2. *G(n)=n.* Se aplica al caso en donde 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 en donde la carga computacional se incrementa más rápidamente que los requerimientos de memoria. De este caso, el modelo de memoria fija dará posiblemente un mayor speedup que el de tiempo fijo.

De este análisis podemos obtener las siguientes conclusiones: la ley de Amdahl y de Gustafson son casos especiales del modelo de memoria fija. 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 1.17) da lugar a un mayor speedup (es decir, $S_n^* \geq S'_n \geq S_n$) y una mejor utilización de los recursos.

En todos los casos, el speedup alcanzable está limitado por el cuello de botella secuencial α y el overhead medio T_0/W, donde T_0 es el overhead total del programa paralelo para la caga escalada y *W* es la carga original. El cuello de botella secuencial es especialmente significativo para aplicaciones de carga fija.

1.5.4 Análisis de la escalabilidad. Función de isoeficiencia

Se han propuesto varias métricas para caracterizar la escalabilidad. Intentan caracterizar lo bien que un sistema paralelo escala mientras se mantiene una eficiencia fija (isoeficiencia), una velocidad fija (isovelocidad) o una utilización fija (isoutilización). En esta sección sólo estudiaremos la función de isoeficiencia, aunque el concepto se puede extender fácilmente para obtener las otras dos métricas.

La eficiencia de cualquier sistema se puede expresar como una función de la carga *W* y del tamaño de la máquina *n*, $E = f(W, n)$. El concepto de isoeficiencia relaciona la carga de una máquina de tamaño *n* necesaria para mantener fija la eficiencia *E* cuando se implementa un algoritmo paralelo en un computador paralelo. Es decir, si fijamos la eficiencia a un valor fijo constante (e.g., 50%) y resolvemos la ecuación de la eficiencia para una carga *W*, la función resultante se denomina *función de isoeficiencia* del sistema.

Para un determinado algoritmo, la carga *W* necesita crecer polinomial o exponencialmente con respecto a *n* para mantener una eficiencia fija. La función de isoeficiencia de los algoritmos paralelos comunes es una función polinomial de *n*. Cuanto menor es la potencia de *n* en la función de isoeficiencia, más escalable es el sistema paralelo.

Cuanto menor es la isoeficiencia, menos carga se necesita para un incremento relativo al incremento del tamaño de la máquina mientras se mantiene la misma eficiencia, así mejor es la escalabilidad del sistema. Por tanto, la isoeficiencia es una herramienta útil para predecir la velocidad de crecimiento de la carga requerida con respecto al incremento en el tamaño de la máquina. Aunque un sistema más escalable, no necesariamente es más rápido.

EJEMPLO

Se va a simular el comportamiento de un algoritmo para los valores de s=8, 16, 32 y 64 (carga de trabajo, W) en arquitecturas hipercubo de *n*=2, 4 y 8 procesadores. Obtenidos los valores de los tiempos de simulación $T(n)$ para cada valor de *s* y *n,* y utilizando los tiempos correspondientes al algoritmo secuencial $T(1) = 8$, 16, 32, y 64, respectivamente para los valores de *s*=8, 16, 32, y 64, se pueden determinar las medidas de ganancia de velocidad (speedup) y eficiencia en las figuras 1.18 y 1.19. Además, a partir de los datos obtenidos se pueden determinar los valores de la isoeficiencia (figura 1.20) que permiten estudiar el grado de escalabilidad del algoritmo.

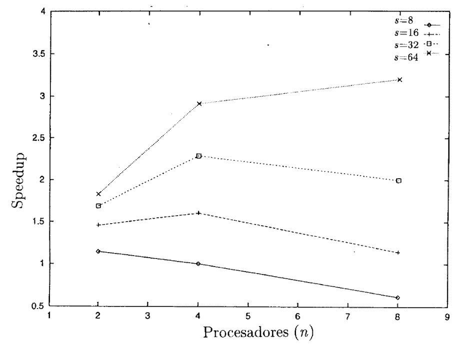


Figura 1.18: Speedup en función del número de procesadores

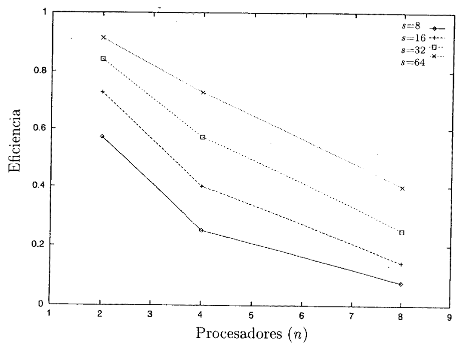


Figura 1.19: Eficiencia en función del número de procesadores

En la figura 1.18 se observa para cada valor de *s* el efecto de involucrar más procesadores en la resolución de un problema dado. Como se puede ver a medida que se incrementa el

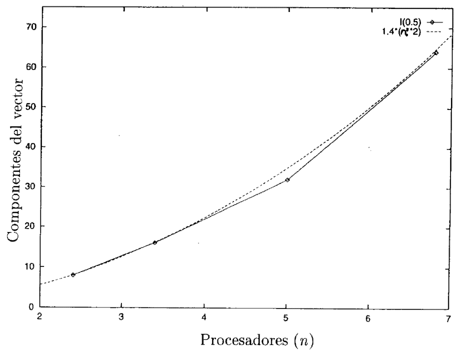


Figura 1.20: Función de Isoeficiencia para E=0.5

número de procesadores, el aumento de la penalización (overhead) debida a la comunicación entre los procesadores y el grado de paralelismo limitado que tiene el algoritmo hace que la ganancia de velocidad no se incremente al mismo ritmo que el valor de n. Así, para el caso de s=8, incrementar procesadores significa reducir la velocidad de resolución del problema. No obstante, a medida que se va incrementado el valor de s se observa que el comportamiento de la ganancia de velocidad va siendo algo mejor. Así, para s=64 se observa que un incremento en el número de procesadores significa un aumento de la ganancia de velocidad, pero el valor de este incremento se va reduciendo a medida que se aumenta n. El comportamiento observado en la figura 1.18 se refleja en las curvas de la eficiencia que proporciona la figura 1.19. A partir de estas se puede concluir que, si se mantiene constante el tamaño del problema (el valor de s), un incremento en el número de procesadores siempre implica una reducción en el uso efectivo que se hace de ellos. No obstante, eso no significa que no tenga sentido incremetar el número de procesadores: es posible sacarles partido si se incrementa al mismo tiempo el tamaño del problema. Esta información se obtiene con la función de isoeficiencia.

Las curvas de isoeficiencia corresponden a los valores que debe tener s (tamaño del problema) para conseguir un valor de eficiencia dado, E, en un multiprocesador con n procesadores. En la figura 1.20 se da la curva de la isoeficiencia para un valor de $E = 0.5$. Para obtenerla se ha partido de las curvas de eficiencia que proporciona la figura 1.19. La curva obtenida se ajusta mediante la función $W = 1.4 \times n^2$, es decir que la curva de isoeficiencia tiene una complejidad cuadrática en n. Si bien la escalabilidad del algoritmo no es tan buena como en el caso de un crecimiento lineal, se puede considerar aceptable.

Ejercicios

1. Un procesador a 40 Mhz se usa para ejecutar un programa benchmark con la siguiente mezcla de instrucciones y ciclos de reloj.

Tipo de instrucción	Contador de instrucciones	Ciclos de reloj
Aritmética entera	45000	1
Transferencia de datos	32000	2
Punto flotante	15000	2
Transferencia del control	8000	2

Determinar el CPI, la velocidad en MIPS y el tiempo de ejecución para este programa

2. Una estación de trabajo usa un procesador a 15 MHz con una velocidad de 10 MIPS para ejecutar un determinado programa mixto. Suponer un retardo de un ciclo de reloj por cada acceso a memoria

(a) ¿Cuál es el CPI efectivo de este computador?

(b) Supón que el procesador se actualiza con un reloj de 30 MHz. Sin embargo la velocidad del subsistema de memoria permanece sin cambiar, y por consiguiente, se necesitan dos ciclos de reloj por acceso a memoria. Si el 30% de las instrucciones requiere un acceso a memoria y un 5% requiere dos accesos a memoria por instrucción, ¿Cuál es el rendimiento del procesador actualizado para el mismo programa mixto?

3. Considerar la ejecución de un código objeto con 200.000 instrucciones en un procesador a 40 MHz. Las mezcla de instrucciones y el número de ciclos (CPI) necesarios para cada tipo de instrucción se dan en la tabla, basados en los resultados de un programa traza

Tipo de instrucción	CPI	Mezcla de instrucciones
Aritméticas y lógicas	1	60%
Load/store con acierto de cache	2	18%
Salto	4	12%
Referencia a memoria con fallo de cache	8	10%

(a) Calcular el CPI medio cuando el programa se ejecuta en un sólo procesador

(b) Calcular la correspondiente velocidad en MIPS basándose en el CPI obtenido en el apartado (a).

4. Los tiempos de ejecución (en segundos) de cuatro programas en tres computadores son los siguientes:

Programas	Computador A	Computador B	Computador C
Programa 1	1	10	20
Programa 2	1000	100	20
Programa 3	500	1000	50
Programa 4	100	800	100

Supón que se ejecutan 100,000,000 instrucciones en cada uno de los programas. Cálcula la velocidad en MIPS de cada programa en cada una de las tres máquinas. Basándose en los resultados, sacar las conclusiones a cerca del rendimiento de los tres computadores.

5. El siguiente segmento de código, formado por seis instrucciones, necesita ejecutarse 64 veces para la evaluación de la expresión aritmética vectorial: $D(I) = A(I) + B(I) \times C(I)$ para $0 \leq I \leq 63$.

```
Load R1, B(I)  /R1←Memoria(α+I)/
Load R2, C(I)  /R2←Memoria(β+I)/
Multiply R1,R2 /R1←(R1)×(R2)/
Load R3, A(I)  /R3←Memoria(γ+I)/
Add R3,R1      /R3←(R3)+(R1)/
Store D(I),R3  /Memoria(θ+I)←(R3)/
```

donde R1, R2, y R3 son registros de la CPU, (R1) es el contenido de R1, α, β, γ y θ son las direcciones de comienzo de memoria de los arrays B(I), C(I), A(I) y D(I) respectivamente. Asume cuatro ciclos de reloj para cada Load o Store, dos ciclos para el Add, y ocho ciclos para la multiplicación en un uniprocesador o en un sólo PE en una máquina SIMD.

(a) Calcula el número total de ciclos de CPU necesarios para ejecutar repetidamente durante 64 veces el código anterior en un computador SISD, ignorando todos los otros retardos de tiempo.

(b) Considerar el uso de un computador SIMD con 64 PEs para ejecutar el código anterior en seis instrucciones vectoriales sincronizadas sobre vectores de datos de 64 componentes, dirigidos por la misma velocidad del reloj. Cálcula el tiempo total de ejecución en la máquina SIMD, ignorando el resto de retardos.

(c) ¿Cuál es el speedup del computador SIMD sobre el computador SISD?.

6. Considerar la ejecución paralela del programa del ejercicio 3 en un sistema de 4 procesadores con memoria compartida. El programa se puede dividir en cuatro partes iguales (50.000 cada una) para una ejecución balanceada por los cuatro procesadores. Debido a la necesidad de sincronización entre las cuatro partes paralelas, se tienen que añadir 5000 instrucciones extras a cada parte.

Asumir la misma mezcla de instrucciones que el ejercicio 3 para cada parte del programa dividido. El CPI para las instrucciones de referencia a memoria (con fallo de cache) se ha incrementado de 8 a 12 ciclos debido a las contenciones. El CPI para los restantes tipos de instrucciones no cambia.

(a) Repetir el apartado (a) del ejercicio 3 cuando el programa se ejecuta en el sistema de 4 procesadores

(b) Repetir el apartado (b) del ejercicio 3 cuando el programa se ejecuta en el sistema de 4 procesadores

(c) Calcular el speedup (o ganancia de velocidad) del sistema de 4 procesadores sobre el sistema uniprocesador

(d) Calcular la eficiencia del sistema de 4 procesadodres comparando el speedup obtenido en el apartado (c) con el caso ideal.

7. Sea α el porcentaje de código de un programa que se puede ejecutar simultáneamente por *n* procesadores. Asume que el código restante se debe ejecutar secuencialmente por un sólo procesador. Cada procesador tiene una velocidad de ejecución de *x* MIPS, y se asume que todos los procesadores tienen la misma capacidad.

(a) Obtener una expresión para la velocidad efectiva en MIPS cuando se usa el sistema para la ejecución exclusiva de este programa, en términos de los parámetros *n*, α, y *x*.

(b) Si *n*=16 y *x*=4 MIPS, determinar el valor de α que nos lleva a un rendimiento del sistema de 40 MIPS.

8. Considerar el uso de un computador de memoria compartida de 4 procesadores para la ejecución de un programa mixto. El multiprocesador se puede usar en cuatro modos de ejecución correspondientes al uso activo de uno, dos, tres, y cuatro procesadores. Asume que cada procesador tiene una velocidad de ejecución pico de 5 MIPS.

Sea f_i el porcentaje de tiempo que se estarán usando *i* procesadores en el anterior programa y $f_1 + f_2 + f_3 + f_4 = 1$. Se pueden asumir las velocidades de ejecución R_1, R_2, R_3, y R_4 correspondientes a la distribución $(f_1,\ f_2,\ f_3,\ f_4)$, respectivamente.

(a) Obtener una expresión para mostrar la velocidad de ejecución armónica media *R* del multiprocesador en términos de f_i y R_i para *i*=1,2,3,4. Mostrar también una expresión para el tiempo de ejecución armónico medio *T* en términos de *R*.

(b) ¿Cuál será el valor del tiempo de ejecución armónico medio *T* del anterior programa dado $f_1 = 0.1,\ f_2 = 0.2,\ f_3 = 0.3,\ f_4 = 0.4$, y $R_1 = 4$ MIPS, $R_2 = 8$ MIPS, $R_3 = 11$ MIPS y $R_4 = 15$ MIPS?.

9. El siguiente programa Fortran se ejecuta en un uniprocesador, y la versión paralela se va a ejecutar en un multiprocesador de memoria compartida.

```
L1:    Do 10 I=1,1024
L2:      SUM(I)=0
L3:      Do 20 J=1,I
L4:  20    SUM(I)=SUM(I)+I
L5:  10 Continue
```

Supón que las sentencias L2 y L4 necesitan dos ciclos máquina, incluyendo todas las actividades de la CPU y acceso a memoria. Ignorar el gasto extra causado por el control software del lazo (sentencias L1, L3 y L5) y todos las demás penalizaciones del sistema (overhead).

(a) ¿Cuál es el tiempo total de ejecución del programa en un uniprocesador?

(b) Divide las iteraciones del lazo *I* entre 32 procesadores con la siguiente asignación: El procesador 1 ejecuta las primeras 32 iteraciones (I=1 a 32), el procesador 2 ejecuta las siguientes 32 iteraciones (I=33 a 64) y así. ¿Cuál es el tiempo de ejecución y el speedup comparado con el apartado (a). (Tener en cuenta que la carga de trabajo dada por el lazo *J* no está balanceada entre los procesadores)

(c) Modificar el programa para facilitar una ejecución paralela balanceada de toda la carga de trabajo en los 32 procesadores. Es decir, igual número de sumas asignadas a cada procesador con respecto a ambos lazos.

(d) ¿Cuál es el tiempo de ejecución mínimo resultante de la ejecución paralela balanceada en los 32 procesadores?. ¿Cuál es el nuevo speedup sobre el uniprocesador?.

10. Considerar la ejecución de dos programas benchmark. En la siguiente tabla se da el rendimiento de tres computadores ejecutando estos dos benchmarks:

Benchmark	Millones deFLOPS	Computador 1 (T_1 sg)	Computador 2 (T_2 sg)	Computador 3 (T_3 sg)
Problema 1	100	1	10	20
Problema 2	100	1000	100	20
Tiempo total		1001	110	40

(a) Calcular R_a, R_g, y R_h para cada computador bajo la asunción de pesos iguales, $f_1 = f_2 = 0.5$.

(b) Cuando el benchmark 1 tiene un rendimiento constante $R_1 = 10$ Mflops en los tres computadores, dibujar R_a, R_g, y R_h en función de R_2, la cual varía de 1 a 100 Mflops bajo la asunción de $f_1 = 0.8\, y\, f_2 = 0.2$.

(c) Repetir el apartado (b) para $f_1 = 0.2\, y\, f_2 = 0.8$.

(d) A partir de los resultados de rendimiento anteriores bajo diferentes condiciones, ¿Podrías sacar conclusiones respecto al rendimiento relativo de las tres máquinas?.

11. Dados dos esquemas *A* y *B* de multiplicación de matrices, ambos tienen el mismo número de procesadores. Los tiempos de ejecución paralelos y las eficiencias de estos sistemas se muestran en la tabla. Tened en cuenta que el tiempo secuencial para multiplicar dos matrices es aproximadamente $T_1 = cN^3$ y la eficiencia viene definida por $E = T_1/(nT_n)$.

Esquemas	Tiempo de ejecución paralelo T_n	Eficiencia E
A	$cN^3/n + bN^2/\sqrt{n}$	$E(A) = \frac{1}{1+(b\sqrt{n})/(cN)}$
B	$2cN^3/n + bN^2/(2\sqrt{n})$	$E(B) = \frac{1}{2+(b\sqrt{n})/(2cN)}$

En cada una de las expresiones de tiempo paralelas, el primer término denota el tiempo de cálculo y el segundo el overhead de comunicación. La diferencia es que *B* tiene la mitad de overhead que *A* pero a expensas de doblar el tiempo de cálculo. ¿Qúe sistema es más escalable asumiendo que queremos mantener (1) *E*=1/3 y (2) *E*=1/4 ?. (La carga para un problema de multiplicación de matrices *NxN* es aproximadamente $W = N^3$).

12. Considerar un programa con una carga de trabajo de una unidad W=1 y un cuello de botella secuencial de cero ($\alpha = 0$). Estimar el speedup de tiempo fijo, $S_n^{'}$ para cada una de las siguientes asunciones:

(a) ¿Cuál es la expresión (big-O) del speedup escalado para un overhead $T_0 = O(n^{-0.5})$?. ¿Podrías conseguir un speedup lineal?

(b) ¿Cuáles son las posibles razones para tener un speedup superlineal en aplicaciones de computación paralela reales?

(c) Repetir el apartado (a) para un overhead $T_0 = O(1)$ y comentar los resultados

(d) Repetir el apartado (a) para un overhead $T_0 = O(logn)$ e interpretar el resultado. ¿Es posible conseguir un speedup lineal si el speedup es más que una constante?.

13. Un algoritmo con un grado de paralelismo de 0.9 tarda 200 sg en ejecutarse en un multiprocesador con *p* procesadores. Suponer que el 20% de ese tiempo corresponde a tareas que hay que ejecutar secuencialmente forzosamente, mientras que el 80% restante han estado trabajando los *p* procesadores. ¿Cuánto tiempo tardaría en ejecutarse el algoritmo en un solo procesador?.