Miscelánea Matemática **56** (2013) 41–53 SMM

El reto de las arquitecturas multinúcleo (*multicore*)

Elisa Viso Gurovich
Departamento de Matemáticas,
Facultad de Ciencias, UNAM
elisa@ciencias.unam.mx

Resumen

Si bien Turing precisó el término *computable*, no le dio importancia al término *eficiencia*, aspecto que se volvió central en los años 70. La equivalencia entre distintos modelos de máquinas de Turing se da en términos de computabilidad pero, en general, con un alto costo en eficiencia. El crecimiento en el poderío de las computadoras con un único procesador obedece a la Ley de Moore, pero está llegando a límites físicos. La manera de hacer más eficientes las aplicaciones es haciendo que varias secciones se ejecuten en paralelo. Sin embargo, la conversión de aplicaciones escritas para ser ejecutadas secuencialmente a aplicaciones que se ejecutarán en paralelo obtiene resultados magros, como lo indica la Ley de Amdahl. La ruta a seguir para reducir significativamente el tiempo de ejecución de una aplicación debe ser el diseño, desde cero, de aplicaciones que se ejecuten en paralelo, como lo indica la Ley de Gustafson. Este ha sido el camino elegido para algunas aplicaciones, pero no ha habido el desarrollo suficiente en el pensamiento en paralelo. Por todo lo anterior, no se han explotado del todo las bondades de las arquitecturas multinúcleo.

1. La máquina de Turing y la arquitectura de computadoras

En la década que empieza en 1930 se da una actividad asombrosa en el estudio formal del concepto de algoritmo. Tanto Gödel indirectamente

en 1930, como Church y Turing en 1936 plantean la posibilidad de la existencia o no de algoritmos que den solución a problemas de índole muy variado, entre ellos el de la demostración de fórmulas en un sistema formal.

Turing en particular plantea un modelo matemático de proceso de cadenas, llamado en su honor Máquina de Turing, que sirve para ejecutar algoritmos; partiendo de este modelo y del trabajo de Alonzo Church un poco antes en el mismo año, la tesis Church-Turing propone, sin que hasta el momento se haya podido contradecir, que toda función *computable* debe poder ser calculada en una Máquina de Turing.

Todos los modelos de Máquinas de Turing –multicinta, multipista, determinista, no determinista, de proceso o de decisión– son equivalentes en cuanto a su poder de cómputo –en el terreno de la computabilidad– pero resultan muy distintas en su desempeño –*complejidad*–, el número de pasos que se requieren para resolver un problema, lo que se traduce como el tiempo que va a tardar en dar la solución.

Si bien Turing estuvo involucrado en el diseño y construcción de muchas computadoras, algunas de ellas de propósito especial, el problema de la complejidad de un algoritmo, el tiempo que tarda en ejecutarse en función del número de datos que procesa, no surge formalmente sino hasta la década de 1970. Es en esos años en que la comunidad científica es consciente de que no basta tener un algoritmo para resolver un problema si ese algoritmo no se va a ejecutar en un tiempo *razonable*, siendo razonable un tiempo que sea una función polinomial (o menor) del número de datos, y como *intratable* aquellos algoritmos cuya ejecución lleve un tiempo que sea función exponencial o factorial en el número de datos, pues estas últimas funciones crecen muy rápidamente.

Una solución para reducir la complejidad en la ejecución de los algoritmos es la paralelización de procesos. Claro que esto tiene límites teóricos, pues exigiría la paralelización arbitraria –contar con un número arbitrario de procesadores–, pero en la vida real podemos conseguir un mejor desempeño, dependiendo del problema del que se trate, incrementando de manera acotada el número de procesadores que atienden a un mismo problema.

2. Ley de Moore

En la década de los noventas y hasta aproximadamente 2004, las arquitecturas de computadoras han buscado, sobre todo, una mayor velocidad del procesador, más memoria cache (dentro del procesador), más

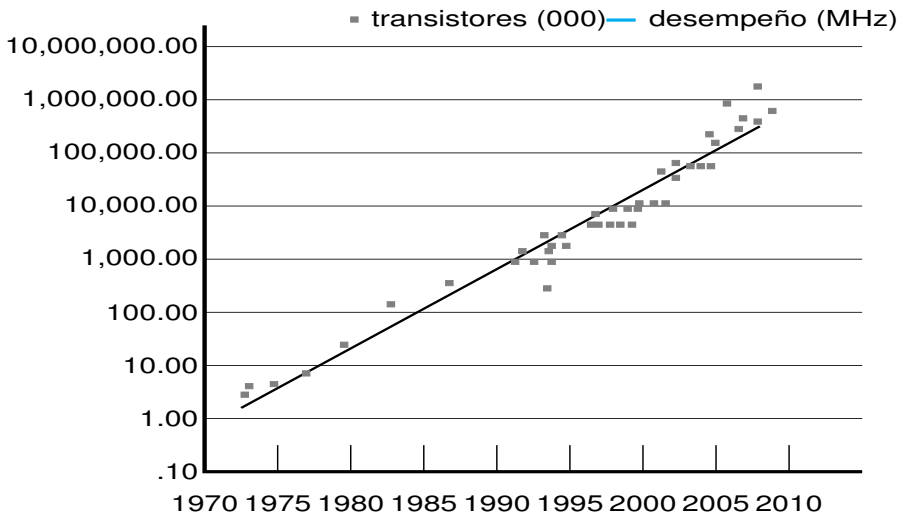

Figura 1: Desempeño de procesadores Intel.

memoria principal y una mayor velocidad en la comunicación entre la memoria y el procesador. La Ley de Moore, publicada en 1965[**5**] y recalibrada en 1975 por su autor, dice que el número de transistores en un circuito integrado se duplica aproximadamente cada dos años. No solo el incremento en el número de transistores ha permitido la construcción de computadoras con, por ejemplo, más memoria o tamaño de palabra mayor, sino que con el aumento simultáneo en la velocidad del procesador, ha permitido un mejor desempeño de las aplicaciones, especialmente aquellas basadas en algoritmos secuenciales. Estos dos aspectos han propiciado durante estas dos décadas el crecimiento de la industria del software, ya que se ha contado con que la próxima generación de procesadores correrá más rápido que la actual.

Para Nathan Paul Myrhrvold, investigador en el Microsoft Research Center, existen varias "leyes del software" que propician el desarrollo en las capacidades de los microprocesadores. Estas leyes son:

i. El software se comporta como o un gas: crece hasta agotar los recursos disponibles.

ii. El software crece hasta que se ve limitado por la Ley de Moore. Crece rápidamente al inicio, aunque limitado finalmente por el software doblega a cualquier procesador.

iii. El crecimiento del software, de cierta manera, hace posible la Ley de Moore. Es el crecimiento del software el que hace que se di-

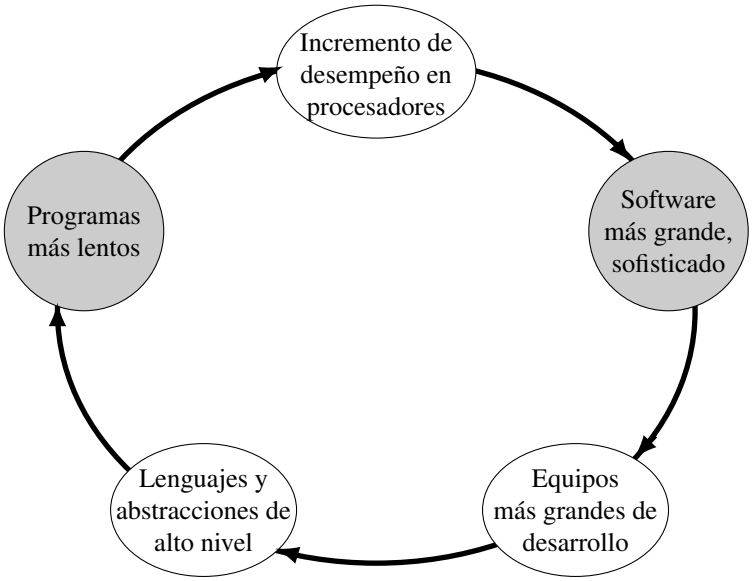

Figura 2: Ciclo de innovación en la industria de la computación.

señe nuevo hardware que, sin aumentar el precio, hace crecer la capacidad del mismo.

IV. El software es como el salario: nunca es demasiado. En cuanto el hardware se vuelve "suficiente", se diseñan algoritmos nuevos, se generan tipos nuevos de aplicaciones y usuarios, y cambian las nociones de lo que es "conveniente" o "necesario".

Esta situación genera el ciclo "virtuoso" del desarrollo del software que se muestra a continuación.

Sin embargo, la energía que se disipa con la velocidad del reloj del procesador y la densidad en los circuitos integrados han llegado a su límite físico; adicionalmente, el paralelismo a nivel de instrucción conseguido en estos procesadores de alta velocidad no ha podido ser explotado adecuadamente. Esto ha hecho que el incremento en el desempeño en procesadores, en la parte superior de la figura 2, ya no puede incluirse en este ciclo.

Si bien la tecnología de semiconductores todavía hace honor a la Ley de Moore, este incremento se usa ahora para poner más procesadores independientes en una misma tableta –más núcleos en el mismo procesador central–, en lugar de tratar de seguir incrementando la velocidad del reloj. Esto, de hecho, acota los incrementos de la velocidad del reloj del procesador, pero el acceso concurrente a la memoria fuera

de la tableta del procesador hace que disminuyan las esperas de acceso. Esta nueva arquitectura también hace que el alambrado dentro del núcleo sea más corto, por lo que los accesos a memoria cache dentro del núcleo son sumamente eficientes. Asimismo, es sabido que dividir para vencer es una estrategia que permite alcanzar el éxito en un plazo más corto. Estas nuevas arquitecturas proporcionan más procesadores (núcleos), pero no más velocidad de cada uno de ellos.

2.1. Investigación relacionada

Los procesos paralelos no son nuevos en el mundo de la programación, pero no se le había dado la importancia que tienen. Realmente, a lo largo de la historia de las computadoras, el software ha ido "persiguiendo" al hardware, en el sentido en que los programadores no han podido explotar a plenitud las arquitecturas que han sido ofrecidas por los fabricantes.

Las arquitecturas multinúcleo han provocado un incremento notable en la investigación relacionada con ellas a partir de 2006–2007. También hay un incremento notable en la investigación en lenguajes de programación y sus procesadores (compiladores e intérpretes). Asimismo se presentan más artículos de investigación en sistemas operativos, tanto en multiproceso simétrico como no simétrico. Todo esto lleva a la comunidad de computación a plantearse el paralelismo o concurrencia de manera seria, ya que esta es la única vía disponible para seguir mejorando los procesos de cómputo.

3. Ley de Amdahl

Si definimos el desempeño o eficiencia de un programa o aplicación como el tiempo que trascurre durante su ejecución, nos interesa cómo mejora este rendimiento si la aplicación es repartida entre varios núcleos o procesadores o, en otras palabras, cómo influye en el desempeño la paralelización de la ejecución de una aplicación.

En 1965 el arquitecto de computadoras Gene Amdahl expuso lo que se conoce como la Ley de Amdahl, que habla sobre la relación que hay entre la ejecución secuencial y paralelizada de una aplicación y predice el factor de aceleración de la aplicación.

Ingenuamente se supone que la relación que existe entre la ejecución de una aplicación frente a la misma aplicación ejecutada en k procesa-

dores es la siguiente:

$$T(A) = \frac{T(1)}{k}$$

Sin embargo, la Ley de Amdahl sostiene que no es así. Amdahl supone un entorno simplificado con las siguientes características:

- El tamaño del programa es fijo.
- El programa tiene una porción S que debe ejecutarse secuencialmente, que no puede ser paralelizada.
- La porción paralelizable del programa la denomina P.
- No considera los costos agregados por la paralelización, como son la intercomunicación entre los segmentos que se paralelizan; la programación necesaria para que inicien los segmentos paralelos; la sincronización entre estos segmentos.

Realmente es el último inciso el que incide de manera importante en el costo de paralelización de procesos. Aún así, el límite impuesto por la velocidad de los procesadores individuales no va a ser resuelto usando varios procesadores trabajando en paralelo, ya que la mejora en el tiempo total del proceso se ve afectado en proporción inversa a la parte del programa que se debe ejecutar serialmente.

Para empeorar la situación, el costo adicional requerido para paralelizar un proceso se agrega al proceso secuencial P, e incrementa el costo del proceso secuencial de cinco a siete veces, aun contando con un procesador adicional. Es claro también que la parte serial del programa no puede repartirse.

Sean T_s el tiempo que se lleva la porción serial del programa; T_p el tiempo que se lleva la porción paralelizable y N el número de procesadores. La Ley de Amdahl nos da una medida de la mejora en el desempeño de un programa en términos del número de procesadores y la forma de particionar el programa:

$$\text{Ley de Amdahl:} \qquad M(N) = \frac{T_s + T_p}{T_s + \left(T_p/N\right)}.$$

Se debe notar que esta fórmula está considerando el ambiente simplificado mencionado con anterioridad. Lo que nos dice esta fórmula es que la mejora en el desempeño tiene un incremento donde únicamente interviene la parte paralelizable del programa.

Sin embargo, debemos considerar el trabajo que implica repartir, coordinar y sincronizar las tareas. Dicho en prosa, la mejora en el

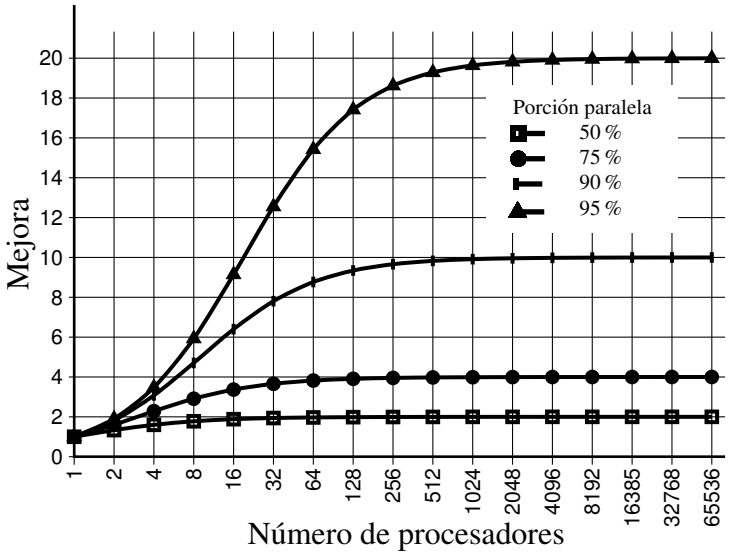

Figura 3: Mejora en desempeño de acuerdo a la Ley de Amdahl.

desempeño de un programa que usa varios procesadores en cómputo paralelo está limitada por la parte serial del programa y la cantidad de trabajo necesaria para coordinación. Con estas consideraciones, y si denotamos con T_{is} el tiempo necesario para coordinación y con T_{ip} el tiempo necesario para iniciar un segmento paralelo (el resto de las variables mantiene su significado), Amdahl propone la siguiente relación:

$$S(N) \leq \frac{T_s + T_p}{T_s + N \times T_{is} + T_p/N + T_{ip}},$$

lo que hace aún más pequeña la mejora en desempeño. En la figura 3 se muestran gráficas de mejora en el desempeño con distintas porciones del programa paralelizables.

Lo que nos dice esta gráfica es que incluso con un programa cuya porción paralelizable es del 95 %, la mejora es notable (casi se duplica la eficiencia) hasta 128 procesadores; más aún, a partir de 512 procesadores prácticamente ya no se consigue mejora. Pero un programa paralelizable al 95 % no es común, menos si le agregamos el costo adicional de lanzamiento, sincronización y coordinación de cada una de las partes a paralelizar. Las líneas correspondientes a porciones paralelizables del 50 y 75 % se acercarían más a la mejora real que podríamos obtener, lo que nos dice que requerimos 64 procesadores para cuadruplicar la mejora en el segundo caso y que no importa el número de procesadores que pongamos para duplicar el desempeño en el primer caso.

En resumen, el límite impuesto por la velocidad de los procesadores individuales al desempeño no se resuelve usando varios procesadores trabajando en paralelo, ya que el tiempo total del proceso se ve afectado en proporción inversa a la parte del programa que debe ejecutarse serialmente.

En la vida real nos enfrentamos también a otros retos, como son:

- El primero que observamos es que el incremento en el límite de velocidad está acotado por factores físicos.
- Por ello, al cumplirse la Ley de Moore, crece el número de núcleos en el procesador sin que se incremente el costo.
- Los nuevos núcleos son más sencillos, baratos y *lentos*.
- Observamos una mejora en la respuesta del trabajo asíncrono que se lleva a cabo en la computadora.
- Se observa que los lenguajes y herramientas no son del todo adecuadas para detectar paralelismo y concurrencia, esto es, convertir un programa escrito bajo el paradigma serial a uno que paralelice lo más posible.
- Por último, tal vez lo más grave, pensar en paralelo no es sencillo y no se está educando lo suficiente en este sentido.

A pesar del pesimismo que impera debido a la Ley de Amdahl, los fabricantes de hardware han hecho avances importantes para ofrecer arquitecturas que permitan la ejecución en paralelo de procesos. Entre los principales apoyos a nivel de la arquitectura mencionamos:

- Entubamiento (*pipelines*), donde la ejecución de instrucciones pretende hacer un uso intensivo de los componentes de la computadora, organizando la ejecución como línea de ensamblado.
- Ejecución fuera de orden y elección de instrucciones, donde se ejecutan aquellas instrucciones listas para ejecutarse, independientemente del orden en el que se encuentren en el programa.
- La microprogramación permite también la ejecución en paralelo al hacer un uso eficiente de los recursos.
- Se agregan a nivel del procesador estructuras de memoria como colas y caches.

- En el código objeto se colocan puntos de verificación (*checkpoints*), se adelanta la ejecución de algunas instrucciones que tienen baja probabilidad de tener que deshacerse y se pospone un tiempo razonable la consumación (*commit*) de las escrituras a disco.

Como preguntas clave para lograr un paralelismo mayor a nivel del hardware tenemos:

- En una arquitectura multinúcleo, ¿cuál es el número óptimo de núcleos? De acuerdo a la Ley de Amdahl, en muchas ocasiones el agregar núcleos o procesadores no trae consigo un mayor nivel de paralelización. Adicionalmente, la respuesta a esta pregunta debe ser de ámbito general y conlleva también la complejidad de cada núcleo, no nada más el número de núcleos.

- Algo que ayuda mucho a mejorar el desempeño de un programa cualquiera es acelerar los accesos a memoria, lo que se hace mediante memorias cache, que se encuentran dentro de la misma tableta que el procesador y por lo tanto son de rápido acceso. Si la computadora va a tener caches compartidos por todos los núcleos, ¿de qué nivel deben ser –que tan cerca del procesador–? ¿Qué tan grandes deben ser los caches? ¿Cuántos bancos o niveles de caches son convenientes?

- La interfaz con la memoria principal (externa a la tableta) ¿con cuántos bancos debe contar?

- Por último, ¿cómo se conectan los componentes dentro de la tableta? ¿Con un bus? ¿Con interruptores? ¿Con alguna topología particular?

Es claro que en este caso casi siempre más es mejor. Sin embargo, como lo muestra la Ley de Amdahl, hay un punto de corte donde ya no se consiguen beneficios al incrementar los costos. Por lo tanto hay que encontrar un punto en el que la relación costo/beneficio (y la implementación física) permita obtener el mayor paralelismo posible.

4. La ley de Gustafson

Pero si el mundo del software se comporta de esta manera, no veríamos avances en las aplicaciones en cuanto a velocidad, contexto o número de usuarios, y estos avances son una realidad innegable. En 1988 John

L. Gustafson rebate la Ley de Amdahl[**1**], argumentando que en la producción de software los programadores, en general, no trabajan con un programa fijo que tratan de paralelizar, sino que tienden a establecer el tamaño del programa en términos del equipo que tienen disponible para resolverlo en un cierto tiempo. Por lo tanto, si el programador cuenta con equipo más veloz (más paralelo), planteará programas más grandes a ser resueltos en el mismo tiempo.

Si consideramos a P como el número de procesadores, α la fracción no paralelizable del programa y $S(P)$ la mejora en el tiempo de proceso, según Gustafson y Barsis, la función que da la mejora se expresa como:

$$S(P) = P - \alpha(P - 1).$$

En otras palabras, argumenta que el programador podrá hacer uso de todos y cada uno de los procesadores adicionales, dedicando uno exclusivamente para la parte no paralelizable.

4.1. Aplicaciones que explotan el paralelismo en hardware

Aun cuando sostenemos que la arquitectura (o ingeniería) de hardware va más adelantada que la de software, hay varios terrenos en los que la Ley de Gustafson se cumple claramente y podemos ver un uso benéfico del paralelismo. Revisaremos algunos de estos contextos.

4.1.1. Bases de datos

Si hay un terreno en el que la concurrencia y el paralelismo han sentado sus reales es en el de las bases de datos, ya que cuentan con las siguientes características:

- En general, las bases de datos pueden distribuirse en varios bancos o equipos, o pueden organizarse bajo determinados criterios, facilitando así su concurrencia
- Teniendo una base de datos distribuida o clasificada podemos poner a cada núcleo a realizar la búsqueda (o actualización) en el banco o porción que le es asignada.
- La mayoría de las veces son los datos mismos los que controlan la concurrencia sobre ellos mismos; son mecanismos de concurrencia inherentes a los datos.

- Es baja la probabilidad de que más de un proceso intente, en paralelo, modificar el mismo dato.
- Aunque los mecanismos de control sean costosos, es baja la probabilidad de que tengan que entrar en acción.

Dadas estas circunstancias, podemos decir que las bases de datos son las más beneficiadas con el paralelismo, aclarando que, como lo dice la Ley de Gustafson, se están escribiendo programas y aplicaciones nuevas que aprovechan el paralelismo presente en el hardware; los paradigmas de acceso a bases de datos son totalmente distintos que los utilizados en las dos últimas décadas del siglo XX.

4.1.2. Cómputo visual

El cómputo visual no solo se ha visto beneficiado del paralelismo del hardware sino que ha sido uno de los factores más importantes en su desarrollo. Este terreno ha sido fértil en la paralelización y es el impulsor de más núcleos sencillos en la tableta del procesador, al grado de que procesadores que empezaron propuestos para graficación se utilizan hoy en día para propósito general. La mayoría de los equipos para cómputo visual trabajan con alrededor de 256 núcleos en paralelo. Las características que permiten a esta área hacer tan buen uso del paralelismo son, entre otras:

- Requiere muy alta velocidad de cómputo.
- Los cómputos que se hacen son, en general, muy sencillos pero especializados, por lo que pueden microprogramarse o incluirse en la arquitectura de los distintos núcleos.
- Todo el proceso de cómputo visual es en tiempo real.
- Se pueden repartir las áreas entre los procesadores disponibles, ya que los cambios en cada una de ellas es independiente de las demás.
- No hay necesidad de intercomunicación.
- La sincronización es muy sencilla, ya que se puede dar a intervalos uniformes de tiempo.

Aunque para el que está viendo una aplicación visual pareciera que todo está interrelacionado, en realidad no es así, ya que se puede predecir la transformación aislada de cualquiera de las áreas; la sincronización

consiste en conseguir que la ejecución se dé en la secuencia adecuada y con los retrasos adecuados, que, en general, están predeterminados o controlados centralmente.

4.1.3. Aplicaciones científicas

De las primeras áreas en las que se consideró importante (y viable) la paralelización de procesos fue, precisamente, en cómputo científico. Este caso podemos decir que es el opuesto al de cómputo visual, pues los cálculos que se han suelen ser complejos, pero si se pueden repartir, resulta ser el mismo proceso a cada parte involucrada. El reconocimiento de que es fácil paralelizar el cómputo científico es palpable en las construcción de las llamadas supercomputadoras, que tienen procesadores homogéneos encargados de hacer operaciones vectoriales, donde cada elemento del vector se calcula independiente del resto durante cada ciclo de la iteración. Asimismo, dada la nula dependencia entre los elementos del vector, hay poca necesidad de coordinación entre los procesadores. Como los núcleos son simétricos y homogéneos, hay poca necesidad de sincronización ya que el tiempo que se va a tardar cada uno de ellos en realizar su trabajo es idéntico a cualquier otro núcleo en el vector.

5. Conclusiones

Con los límites físicos en cuanto a la velocidad del reloj de los procesadores se rompió el círculo virtuoso de mejoras basadas en un mejor desempeño del procesador. Pensar en paralelo no es algo fácil para los programadores, especialmente aquellos educados en algoritmos secuenciales, por lo que el énfasis en la paralelización debe darse en los compiladores que procesan programas arbitrarios para conseguir paralelización. Desafortunadamente en estos casos la Ley de Amdahl no da mucho espacio para moverse, aunque se pueden conseguir resultados aceptables, sobre todo en lenguajes de propósito específico, donde la Ley de Gustafson es la que se cumple.

Se requiere de investigación en la dirección de hacer algoritmos paralelos que sustituyan a los algoritmos secuenciales, en la medida de lo posible, pues un buen compilador (o intérprete) podrá paralelizar más algo que tiene pocas partes seriales.

En teoría de complejidad computacional, la clase NP (*Nondeterministic Polynomial time*) es aquella que contiene a todos los problemas de decisión donde en tiempo polinomial, una Máquina de Turing

determinista puede decidir si una solución dada al problema es correcta o no. La solución de un problema en NP se obtiene usando una Máquina de Turing no–determinista, donde todos los caminos se van obteniendo simultáneamente. La ejecución de una Máquina de Turing no–determinista se puede representar con árboles, y lo que esta definición nos dice es que cada una de las ramas del árbol puede ser calculada en tiempo polinomial. La clase de complejidad P es aquella que contiene a todos los problemas de decisión que pueden ser resueltos por una Máquina de Turing determinista en tiempo polinomial. Si pudiésemos ejecutar en paralelo todas las ramas de un problema dado NP, el tiempo total para obtener la solución sería el de la rama más larga, o sea polinomial. Traducido a la ejecución de aplicaciones, querría decir que cualquier algoritmo es susceptible de ser paralelizado, acortando su tiempo de ejecución. De ser posible esta paralelización, tendríamos $NP = P$, lo que los teóricos de la computación consideran improbable. Adicionalmente, en términos prácticos, el número de núcleos está fijo y el número de ramas de un problema de decisión puede crecer dinámicamente con el número de datos, lo que haría imposible asignar a cada núcleo una rama del árbol.

Bibliografía

1. G. Amdahl, Afips conference proceedings, en *Validity of the single processor approach to achieving large-scale capabilities*, 1967, 483–485.
2. J. L. Gustafson, Reevaluating amdahl's law, Communications of the ACM, tomo (31) 5, 1988, 532–533.
3. M. D. Hill y M. R. Marty, Amdahl's law in the multicore era, *IEEE Computer* (2008) 33–38.
4. Intel, *Microprocessor quick reference guide*, 2008.
5. G. E. Moore, Cramming more components onto integrated circuits, *Electronics Magazine* (1965) 4.