La supercomputación

Félix García Merayo
Dr. en Informática
Profesor Titular de Universidad-UPM

Muerte, impuestos y procesamiento paralelo: nadie está a favor de ellos, pero son hechos inevitables de la vida.

A. Chalmes y J. Tidmus

UNA DEFINICIÓN DE PARTIDA

El término *supercomputador* se refiere al ordenador perteneciente a la gama más alta en cuanto a rendimiento, capacidad de almacenamiento y eficiencia posibles, de todos aquellos disponibles en un momento determinado del tiempo. Por lo tanto, el atributo *super* debe considerarse como una característica definitoria de un computador que dispone de la más alta tecnología existente en el mercado.

El término es relativo. Hace algunos años, las agencias gubernamentales americanas exigían de sus proveedores para clasificar un sistema de computación como superordenador, una potencia de cálculo de, al menos, 10^8 operaciones en coma flotante por segundo, es decir, 100 Megaflops. Hoy día, la exigencia se ha elevado hasta límites de varios Gigaflops (10^9) e incluso varios Teraflops (10^{12}) .

El vocablo sajón *flops* es una abreviatura de *floating point operation per second*, es decir, operaciones ejecutadas en aritmética de coma flotante en un segundo. Se trata de operaciones con números reales a diferencia de otras ejecutadas por el ordenador en las que solo intervienen números enteros. Como media, cada operación con números reales viene a emplear tres veces más tiempo que otra ejecutada con solo enteros.

Más adelante descubriremos cómo pueden alcanzarse esas potencias de cálculo y convertir así un computador en una máquina tan rápida.

Debido a la tecnología actual empleada en el diseño y construcción de los supercomputadores, hoy día se utiliza mucho más el término *computador paralelo* para designar esas mismas máquinas. Un computador paralelo es un conjunto de elementos de proceso, procesadores, que se comunican entre sí cooperando todos ellos en la resolución de un mismo o único problema de gran tamaño. Hemos trascrito la definición de Almansi y Gottlieb, año 1989.

Esta definición tan breve encierra sin embargo muchas cuestiones:

1. **Conjunto de elementos de proceso**: Cuántos; podrá aumentarse su número indefinidamente; cuál será la potencia de cada uno de ellos; qué tipo y tamaño debe tener la memoria principal; cuál es la organización de tal sistema; cómo se realizará la entrada y salida de la información.

2. **Capaces de comunicarse entre sí**: cómo se conectan entre sí los elementos de cálculo; qué tipo de información son capaces de intercambiar entre sí; qué protocolo utilizan para lograr ese intercambio.
3. **Cooperación**: cómo sincronizan sus labores los distintos elementos de proceso; cuál es el grado de autonomía de cada uno de ellos; cómo los considera el sistema operativo del sistema total; qué tipo de operativa hace que puedan secuenciarse y coordinarse las acciones llevadas a cabo por los diferentes procesadores.
4. **Objetivo, resolución de un único problema**: cuáles son los problemas a resolver por el paralelismo; qué modelo de cálculo se emplea; cómo se eligen los algoritmos a programar posteriormente; cuál es el grado de especialización de la máquina a utilizar frente a un problema dado; cuáles son los lenguajes de programación que deben emplearse.

La respuesta completa a todos estos interrogantes, y otros más no planteados, constituye todo un curso de especialización sobre arquitecturas paralelas que debe tomar un ingeniero informático para llegar a comprender el funcionamiento y utilización de estos computadores. Eso significa que las cuestiones exceden del entorno de esta introducción. No obstante, desvelaremos, en cierto grado, algunas de las respuestas a las cuestiones planteadas anteriormente.

Eso sí; contemplando todos esos interrogantes, deducimos que casi se trata de reinventar la informática en lo relativo a máquinas, conceptos, lenguajes de programación, modelos de computación, etc.

Como consecuencia del segundo término definido, computador paralelo, se habla de *computación paralela* al referirnos a la ejecución de problemas en esas máquinas.

UN POCO DE HISTORIA COMO INTRODUCCIÓN

El computador y la Ciencia Computacional, junto con su avanzado equipo tecnológico en uso, constituye lo que modernamente denominamos Computación de Altas Prestaciones, *High Performance Computing* o simplemente *HPC*. A estas siglas se le van añadiendo otras a medida que el tiempo decide la utilización del computador para otros fines, como pueden ser las comunicaciones: nace *HPCN*, *High Performance Computing and Networking*. Con este tipo de tecnología se puede solucionar un amplio espectro de aplicaciones tanto técnicas como relativas a la investigación científica e industrial. HPC, HPCN y otras, en el fondo son formas distintas y, a veces sinónimas, de designar la *supercomputación*.

La arquitectura de los supercomputadores modernos está marcada por un rasgo dominante que ya hemos citado: la utilización de varios procesadores en una misma máquina. Esta es una de las formas, aunque no la única, de conseguir altas velocidades de cálculo. Ello da lugar a las denominadas, de forma genérica, *máquinas paralelas*, máquinas con las que pueden alcanzarse rendimientos muy altos sobrepasando en muchos órdenes de magnitud al de los computadores usuales, rendimientos necesarios para ejecutar de forma eficiente aplicaciones como las relativas a la física nuclear, meteorología, astrofísica, inteligencia artificial, química computacional, *data mining*, y otras muchas de las que se hablará en otra ocasión.

La arquitectura del computador tiene unos cincuenta años de historia; casi al mismo tiempo y fundamentada en este dispositivo automático y electrónico, nace la ciencia informática. Hagamos un poco de resumen histórico.

- El modelo clásico de arquitectura de ordenador de programa almacenado es el definido por J. von Neumann y aparecido en los años 50. Figura 1.

Figura 1.Esquema de una arquitectura Von Neumann

Al mismo tiempo, y en esa misma década, asistimos a los primeros ensayos de construcción de máquinas paralelas; es ya en los años 60 y 70 cuando realmente aparecerán las primeras de ellas, aunque con muy poco éxito comercial. Es el caso del *ILLIAC IV* desarrollado por la Universidad de Illinois en los 60. Poseía 64 elementos de cálculo disponiendo cada uno de su propia

memoria y estaban todos ellos físicamente unidos por medio de una red de interconexión propia del sistema.

- En la década de los 60 se crea una auténtica industria informática alrededor de los ordenadores de la época evolucionados respecto a los primitivos y ello propiciado por el progreso de la electrónica en cuanto al tamaño de los circuitos, su coste de fabricación y su rapidez. La mayor parte de las máquinas paralelas de esta época, máquinas *Cyber*, se conocían como calculadores *vectoriales* y poseían un solo procesador.

- Los sistemas paralelos multiprocesador aparecen al final de la década de los 70. Podemos hablar de varias categorías.

  1. **Máquinas vectoriales multiprocesador**, desarrollo evolucionado de las vectoriales monoprocesador aparecidas en la década anterior. Estaban constituidas por procesadores muy potentes y capaces de calcular a razón de 300 Mflops. Los procesadores de estos sistemas disponían de una memoria compartida entre todos ellos. Los más modernos de esta familia llegaron a alcanzan potencias de cálculo de hasta 10 Gflops. Debido a su coste de producción para aumentar su capacidad de cálculo, y por lo tanto una alta relación coste/rendimiento, hace varios años que estos sistemas han dejado de producirse.

  2. **Multiprocesadores de memoria distribuida**, máquinas caracterizadas por el empleo de procesadores clásicos pero en un gran número, desde 16 hasta un máximo de 1024. Cada procesador dispone de su memoria local y la comunicación entre los distintos procesadores se hace mediante el paso de mensajes entre ellos a través de una red de comunicaciones. Cada procesador ejecuta su propio programa almacenado, lo que significa que, en un cierto momento y en un mismo sistema de computación, se están llevando a cabo cálculos distintos. Ejemplos de esta familia: los *transputer*, del inglés *TRANSistor comPUTER*, *iPSC*, *FPS*, los *T3D* y el *SP-2* de la compañía IBM.

  3. Por último citaremos las **máquinas síncronas** descendientes directas de la ya citada ILLIAC IV de las que la más conocida fue la *Connection Machine 2*. Hace años desaparecieron del mercado debido a que solo podían resolver problemas específicos; no eran sistemas de propósito general. Contenían un elevado número de elementos de cálculo o procesadores: desde 4096 hasta 65536, pero cada uno de ellos solo podía tratar entre 1 y 4 bits (dígitos binarios). En un momento determinado se ejecutaba una misma instrucción en todos ellos, pero cada una se alimentaba de datos distintos que se encontraban almacenados en sus memorias locales.

FACTOR DECISIVO: EL RENDIMIENTO

La arquitectura de los computadores, su tecnología y las aplicaciones que resuelven, son entidades que trabajan juntas y poseen grandes interacciones. La arquitectura de las máquinas paralelas no es una excepción. En este caso y al escenario del diseño, como es el número de procesadores, hay que añadir una nueva coordenada bajo la cual gira todo el proyecto: alcanzar un rendimiento elevado a un coste aceptable. Qué rendimiento adicional puede conseguirse y a qué coste, es algo que depende de muchos factores.

Para comprender mejor esas interacciones, consideremos las características del rendimiento de diferentes procesadores. El gráfico de la figura 2 traído aquí de Hennesy y Joupi, 1991, y formalizado para explicar los tremendos cambios que han tenido lugar en la industria de los computadores, muestra el crecimiento del rendimiento de los procesadores utilizados en cuatro familias distintas de ordenadores, las comerciales existentes desde la segunda mitad de los años setenta, y en un cierto intervalo de tiempo comprendido entre los años 1965 y 1995: solo treinta años. Las líneas cortadas representan las tendencias en 1995. El gráfico sugiere varias conclusiones.

1. El rendimiento de los microprocesadores se ha ido incrementando a razón de un 50% cada año trascurrido desde mitad de los ochenta.

2. El rendimiento de la mayor parte de los grandes sistemas, *mainframes*, y de los supercomputadores desde su aparición en los setenta, se ha incrementado en solo un 25% cada año.

3. Como consecuencia, el microprocesador es el elemento más conveniente para hacerlo intervenir como pieza clave en el diseño y construcción

de las máquinas paralelas de hoy día. Y así se ha hecho. Las ventajas de emplear procesadores pequeños, baratos, de baja potencia y producidos de forma masiva, es la clave que proporciona buenos rendimientos a bajo coste.

4. Los micros con tecnología CMOS, *Complementary Metal Oxide Semiconductor*, siguen las predicciones hechas, mientras que los grandes sistemas y los supercomputadores tal como fueron diseñados en un principio, han ido pasando por tremendas crisis de mercado.

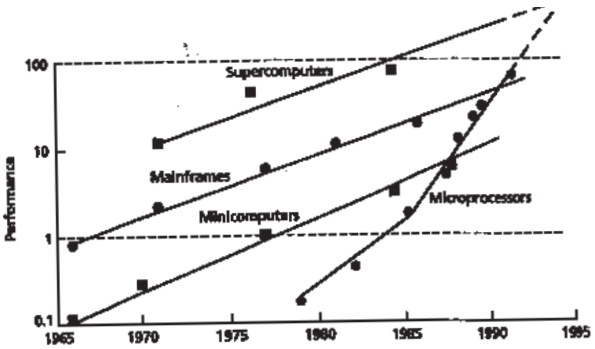

Figura 2. Tendencia del rendimiento de las distintas familias de ordenadores

Como se aprecia en el gráfico de la Figura 2, al final de los noventa acontece una interesante situación al encontrarnos en el punto de convergencia de todas las tendencias; es el momento en que el microprocesador, constituido por un solo chip, pasa a dominar todos los sectores de la computación y cuando el procesamiento paralelo pasa a liderar muchas áreas de la corriente de la computación. No obstante, este cambio aconseja ser cautos de cómo extrapolar el futuro, es decir, los rendimientos esperados.

La demanda de un rendimiento cada vez mayor de las aplicaciones ejecutadas por el ordenador es un aspecto, como hemos anticipado, obligado en la computación. Los avances del hardware posibilita nuevas funcionalidades de las aplicaciones que, a su vez, demandan nuevas arquitecturas. Los campos más tradicionales han sido los correspondientes a la ingeniería de todo tipo, pero sobre todo, lo fueron aplicaciones como predicción del tiempo, modelado del plasma, diseño farmacéutico, química cuántica, etc. El siguiente gráfico, Figura 3, en el que se han dejado los titulares en inglés, (Fuente: U.S. High Performance Computing and Communications Program) muestra ciertas aplicaciones en cuanto a capacidad de memoria y rendimiento computacional necesarios. Es obvio que no podrían ejecutarse fuera del entorno propio de la supercomputación. Debido al crecimiento exponencial, tanto de los rendimientos como de la capacidad de almacenamiento, ambos ejes del gráfico siguen una corriente paralela a la coordenada *tiempo*.

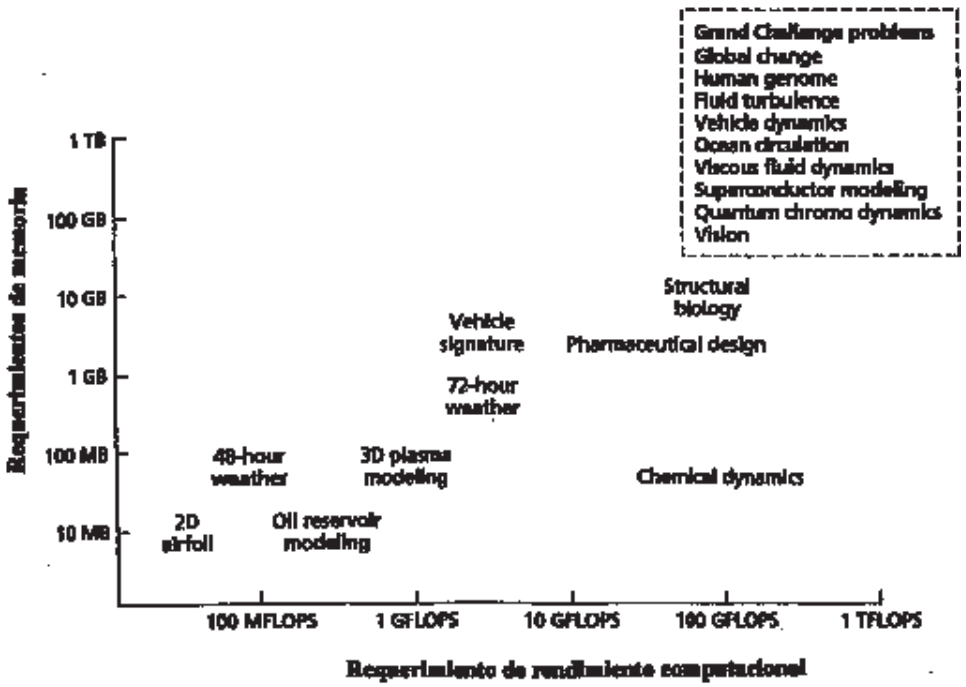

Figura 3. Requerimientos de las aplicaciones

En la pasada década, el programa americano mencionado *High Performance Computing and Communications* ya identificaba como aplicaciones *gran reto* para la supercomputación, las siguientes: cambio climático global en amplios períodos de tiempo, genoma humano, turbulencia de fluidos, dinámica de vehículos, circulación del océano, dinámica de fluidos viscosos, modelado de superconductores, cromo-dinámica cuántica, visión artificial.

Aunque se consiguieran incrementos notorios en el rendimiento del procesador, nunca serían suficientes para cubrir las necesidades computacionales requeridas por las aplicaciones a las que acabamos de referirnos y de las que puedan aparecer en el futuro próximo. Es decir, para poder ejecutarlas eficientemente en tiempos útiles, se hace necesaria la utilización de las arquitecturas masivamente paralelas que se han convertido en imprescindibles para la computación científica, lo cual hace, como consecuencia, que también ciertas industrias como la del petróleo, automóvil, aeronáutica, farmacéutica, ..., basen el proceso de sus aplicaciones anejas en este tipo de supermáquinas. Además, la mayoría de esas aplicaciones demandan la visualización de sus resultados lo cual, de nuevo, hace que el empleo del paralelismo sea un recurso imprescindible.

La demanda constante de un rendimiento creciente es una consecuencia natural de la propia actividad de la modelización. Por ejemplo, en el diseño de circuitos electrónicos mediante herramientas CAD, *Computer Aided Design*, ocurre que a medida que el total de dispositivos dentro de un mismo chip se incrementa y se hace necesario un mayor nivel de funcionalidad entre ellos, cada prueba o *test* consume un gran número de ciclos de reloj del procesador empleado para esa modelización. A esto hay que añadir el nivel de confianza requerido puesto que el costo de fabricación de esos dispositivos electrónicos es elevado. Todo lo anterior nos indica que el efecto acumulativo es que la demanda computacional para la verificación del diseño de cada nueva generación de chips se incrementa, incluso, en mayor proporción que el propio rendimiento del microprocesador a fabricar.

Añadamos ahora una reseña en lo concerniente a la computación de tipo comercial y de gestión. Este tipo de aplicaciones también necesitan, desde hace pocos años, en concreto, desde la mitad de la década de los sesenta, de las arquitecturas multiprocesador para su ejecución. Aunque la escala de este paralelismo no sea tan pronunciada como en el caso de la científicas, su empleo es aún más universal y no solo utilizado por minorías. Basta con considerar las aplicaciones que deben soportar un número elevado de transacciones *on-line*.

En el siguiente gráfico, Figura 4, que es una consecuencia del informe *Transaction Processing Performance Council*, *TPPC*, del año 1996, fuente de la cual se han extraído los valores, se muestra el *throughput*, parámetro que, en este caso, significa el total de transacciones por minuto o *tpm*, frente al total de procesadores para conseguirlo. Se indican esos valores para

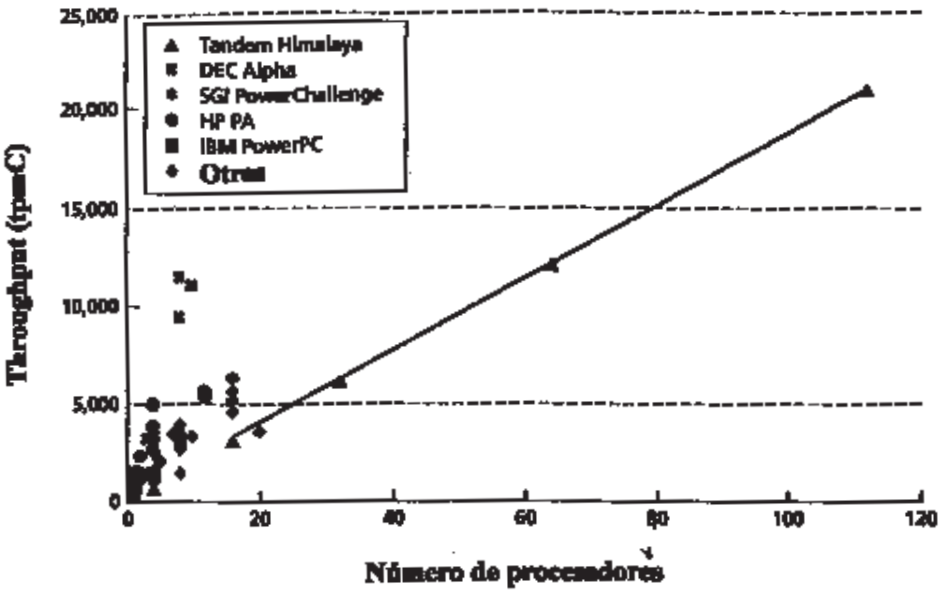

Figura 4. Tpm respecto al número de procesadores del sistema

cinco máquinas de diferentes marcas y fabricantes líderes del mercado.

En todos los casos reseñados en el gráfico hay un factor común: el uso de arquitecturas paralelas es importante y todos los fabricantes están capacitados para ofrecer a sus clientes sistemas paralelos multiprocesador de rendimientos sustancialmente superiores a sus productos monoprocesador.

La arquitectura de los computadores pone su potencial tecnológico a disposición del rendimiento y capacidad de las máquinas y las dos vías para que el aumento de recursos, más transistores, mejore ese rendimiento son el paralelismo y la localidad. Eso significa, una vez más y para resumirlo, uso de multiprocesadores y de memorias masivas de acceso directo.

EJEMPLOS SIMPLES PARA CONCEPTOS COMPLEJOS

Vamos a ilustrar una serie de conceptos relativos a la supercomputación proponiendo un ejemplo muy sencillo que encontramos en un texto de los muchos existentes en el mercado para el estudio, a nivel universitario, del paralelismo, *Practical Parallel Processing*, editado por Thomson en 1996. En concreto, se trata del vaciado manual de una pequeña piscina utilizando para ello dos tipos de recursos, personas y cubos.

En el ejemplo propuesto el trabajo total, *job* en inglés, puede subdividirse en la tarea repetida, *task*, de sacar un cubo de agua de forma sucesiva y por una sola persona. Se muestra en la figura adjunta.

Este hecho definiría el concepto computacional de trabajo **en serie** o **secuencial que llevaría a cabo** un **sistema monoprocesador**: una tarea no comienza mientras no haya finalizado la anterior. El trabajo lo ejecuta una sola persona y terminará una vez finalizadas todas las tareas posibles, vaciado completo, en un cierto tiempo t_s.

Al definir anteriormente el procesamiento paralelo, nos hemos referido a la **cooperación**. Esa cooperación entre los sujetos agentes será siempre necesaria para alcanzar mejor la solución de un problema, aunque se trate simplemente de un acuerdo de distribución del trabajo.

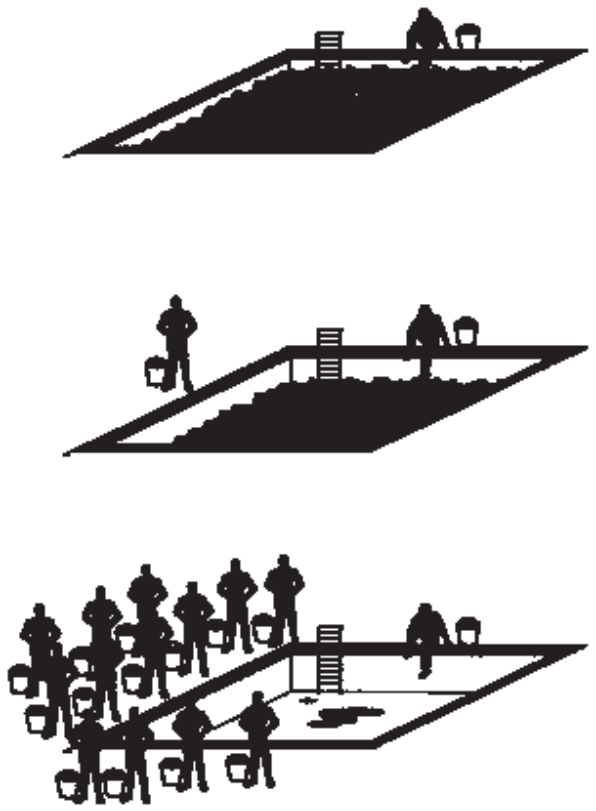

Figura 5. Vaciado de una piscina

El vaciado puede hacerse más rápido, puede acelerarse, si en lugar de emplear una sola persona lo ejecutan, colaborando, dos o más, tal como representa el dibujo. Teóricamente, dos personas tardarán la mitad de tiempo; varias lograrán reducir la duración total a una fracción del tiempo original t_s. En cualquiera de estos casos, y desde el punto de vista computacional, diríamos que ejecutamos las tareas de forma **paralela**, o mediante **un procesamiento paralelo**, en un tiempo total t_p.

¿Cómo mediremos la ganancia en velocidad de actuar secuencialmente o de forma paralela?. Esa medición conduce a un concepto nuevo denominado **aceleración *s***, en inglés *speed-up*, cuya expresión es

$$s = t_s / t_p$$

Así, si una sola persona emplea una hora y dos personas lo finalizan en media hora, la aceleración o ganancia en velocidad, será $s = 1 / 0{,}5 = 2$, es decir, el trabajo total se ejecutará *dos* veces más rápido, lo que equivale a concluir que con un proceso cooperativista y paralelo el rendimiento alcanzado es dos veces superior al obtenido trabajando de manera secuencial por un solo agente o recurso.

Podríamos también desear conocer cuánto contribuye cada persona o agente a alcanzar esa aceleración en el vaciado, para lo que sería suficiente con dividir la aceleración total alcanzada *s* entre el total de personas *p*. El cociente mide el parámetro denominado **eficiencia *E***:

$$E = s / p$$

Pasando del ejemplo al propio sistema paralelo, su eficiencia se obtendría dividiendo la aceleración por el total de procesadores empleados en el sistema.

Pero en la realidad las cosas no ocurren con tanta facilidad y *limpieza*; existen algunas limitaciones físicas que contribuyen negativamente en esta hipotética situación.

Por ejemplo, el vaciado precisa de un cierto orden entre las personas o agentes para que no se produzcan interferencias entre ellos al trabajar en un mismo medio, en una misma piscina . Lo mismo ocurriría si dos o más agentes tuvieran que utilizar el mismo cubo. En el argot computacional, esas interferencias, reciben el nombre de **colisiones**. El tiempo necesario para evitarlas y lograr un orden adecuado, supone una sobrecarga, un *overhead*, que hará que se retrase el final del trabajo total y que se emplee más tiempo del previsto. Esto también ocurre en los procesamientos paralelos reales cuando, por ejemplo, dos procesadores distintos, que constituyen el sistema paralelo, demanden un mismo dato de la memoria principal. Será necesario establecer un orden de prioridades para alimentar uno de los procesadores primero y después el otro.

Para el vaciado de la piscina no se puede emplear cualquier número de personas; ese número tiene un límite máximo, sobrepasado el cual, las personas se interferirían mutuamente debido a las limitaciones propias del espacio, de las puertas por donde habrán de pasar o por otras condiciones físicas. La solución, y con ello el tiempo de finalización, lejos de mejorar, empeoraría. Si suponemos que para vaciar la piscina de una sola vez fueran necesarios 10000 operarios, cada uno con su respectivo cubo, está claro que el proceso se arruinaría debido al desorden que tal situación proporcionaría al tratar de pasar todos ellos por una única puerta de salida, por ejemplo. Se obtendría un rendimiento peor que empleando solo el personal adecuado debido a esos **cuellos de botella**, *bottleneck*. Ese límite superior proporciona una idea de lo adecuado que resulta el aplicar procesamiento paralelo a un determinado problema. Computacionalmente, ese límite mide la **escalabilidad** del proceso paralelo. Un proceso se dice que es muy escalable cuando se adecua bien a ser resuelto por técnicas de paralelismo.

Los retrasos causados por el cuello de botella pueden ser tan importantes que el tiempo empleado para el vaciado con un número tan alto de trabajadores superaría al que se obtiene empleando una sola persona.

Otro factor que puede empeorar el rendimiento de una solución paralela son las **dependencias**. Cada trabajador debe actuar siguiendo un orden físico: tomar el cubo respectivo, bajar a la piscina, llenarlo, salir de la misma y llevar el cubo al lugar de vaciado. El resto de operarios harán lo mismo y también en ese orden. Ningún agente debe sumergirse en la piscina si antes no dispone del respectivo cubo; no debe salirse de la piscina con el cubo sin llenarlo completamente. Eso significa una dependencia. Las dependencias dividen un problema total en un determinado número de etapas distintas. La solución paralela de cada etapa ha de completarse antes de comenzar la etapa siguiente. Esto también ocurre en la computación: una simple instrucción de suma de dos cantidades no podrá ejecutarse mientras el procesador no tenga en sus registros, *no conozca*, las dos cantidades a sumar.

Las dependencias dentro de un problema pueden ser tan grandes y numerosas que hagan que no sea posible el procesamiento paralelo de las mismas. Un problema estrictamente secuencial consta de un número determinado de etapas, cada una constituida por una sola tarea y de tal forma que cada tarea depende de la anterior. Pensemos en una construcción de juguete *Lego* a base de bloques de plástico que se ensamblan unos con otros. Para construir con ellos una torre se requiere un orden ineludiblemente secuencial: cada pieza encima de la anterior hasta colocar la más alta. Esta situación es la contraria a una totalmente libre de dependencias, como sería colocar las piezas una al lado de la otra encima de la mesa, en cuyo caso el orden para acabar todas las tareas es indiferente, aunque, eso sí, exista un cierto grado de cooperación.

Los efectos de las dependencias pueden aminorarse utilizando la técnica de **segmentación**, en inglés *pipelinning*, técnica solo utilizable cuando un proceso, que consta de varias etapas distintas, hay que repetirlo varias veces. Toda línea de montaje en un taller de producción constituye un ejemplo de proceso segmentado. La Figura 6 muestra un *pipelinning* para el ensamblaje de coches. El esquema está muy simplificado. Solo

existen cuatro tareas dependientes en cada proceso: fabricación del chasis, ensamblaje de las partes motrices, colocación de ruedas e instalación del techo.

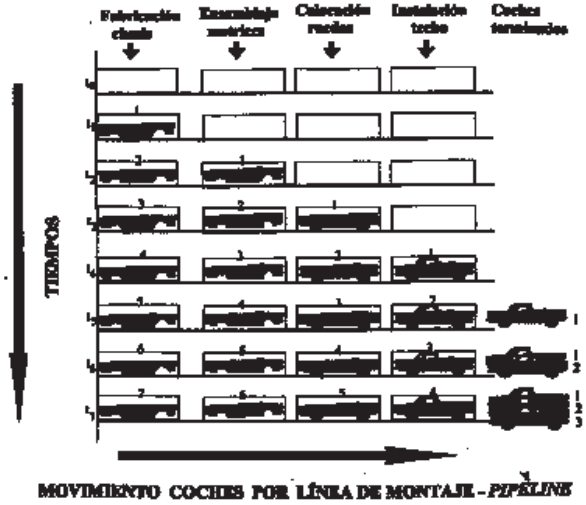

Figura 6. Línea de montaje para ensamblar automóviles

Puede observarse como entra el primer chasis en la línea a través de la etapa primera e irá pasando, sucesivamente, por las tres etapas restantes hasta obtener el producto terminado. Podemos suponer que cada tarea emplea una unidad de tiempo. Finalizada una determinada tarea, el proceso coloca el coche a montar en la etapa siguiente, con lo que esa etapa queda libre para dedicarse a otro coche procedente de la etapa anterior. Después de cuatro unidades de tiempo, cuatro etapas, se libera un coche terminado. Si una de las etapas empleara más de una unidad de tiempo, las etapas que siguen deberán estar detenidas hasta que les llegue producto de esa más lenta.

La segmentación se emplea también en la computación y las máquinas vectoriales, a las que de alguna manera ya nos hemos referido, trabajan bajo esta filosofía. Por ejemplo, se puede segmentar el proceso de una suma de cantidades. Las cuatro etapas, también de forma simplificada, podrían ser: lectura de memoria del primer dato, lectura del segundo sumando, ejecución de la suma propiamente dicha y liberación a la memoria del resultado obtenido en el procesador.

El vaciado de la piscina requiere un cierto grado de **control**. Por ejemplo, podría solicitarse la colaboración de una o varias personas para planificar y controlar tanto la recogida de cubos de vaciado como la puerta o puertas de salida. Con ello se mejorarían los tiempos y, por lo tanto, el rendimiento ya que se evitarían accesos simultáneos.

Las soluciones paralelas de un problema también requieren un determinado control, control que puede ser tan simple como el necesario para determinar lo que constituye una tarea o tan complicado como el requerido para que las distintas etapas produzcan de forma eficiente la solución completa del trabajo total.

La Figura 7 que sigue muestra esquemáticamente la ejecución de un problema de forma secuencial y de forma paralela. En la forma secuencial (a), la computación se aplica al problema a resolver para producir los resultados deseados. La solución paralela controlada (b), realiza una ejecución paralela del mismo resultado anterior a través de cuatro etapas:

1. División del problema total en un cierto número de subproblemas, **SPi**.
2. Aplicación del procesamiento paralelo para lograr que cada subproblema se ejecute a la vez que los restantes para producir los respectivos subresultados, **SRi**.
3. Reunir los resultados de todos los subproblemas ejecutados independientemente para conseguir los resultados finales.

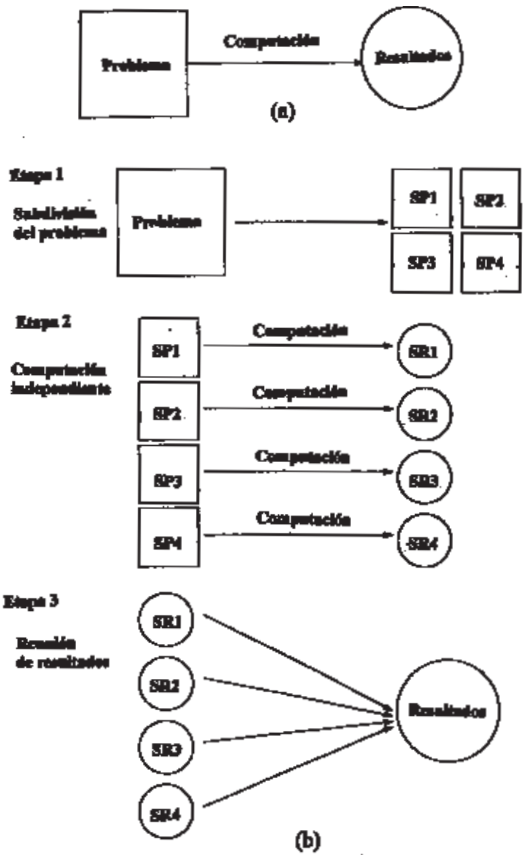

Figura 7. Controles requeridos en un proceso secuencial (a) y en un proceso paralelo (b)

¿Dónde se hace necesario el control en este proceso paralelo? En las etapas 1 y 3 para distribuir el problema entre los agentes de que disponemos, que son cuatro, y posteriormente para reunir los resultados parciales que esos agentes han obtenido por separado y de forma independiente.

Los agentes encargados de ejecutar una computación colaborando entre sí para conseguir la solución de un problema utilizando una máquina paralela se conocen con el nombre de **elementos de proceso**, **PE**, del inglés *Processing Element*. En concreto, un PE consta de: procesador propiamente dicho, una o más tareas en las que trabajar y el software necesario para lograr la cooperación con los otros PE. Por último, un sistema paralelo está constituido por dos o más PE.

Finalmente, nos referiremos a dos conceptos usuales de la supercomputación: el **grano** y el **grado** del paralelismo. Volvamos a la piscina. Recordemos que al hablar de dependencia hemos hecho la hipótesis de que cada agente realiza cinco acciones sucesivamente. Ese número es el grano del paralelismo llevado a cabo en el vaciado: es el total de operaciones realizadas por cada persona. Por otra parte, si suponemos que se dispone de 20 personas para el vaciado en paralelo de la piscina, esa cantidad representaría el grado del paralelismo: es el total de acciones realizadas al mismo tiempo.

LAS CUATRO FAMILIAS DE FLYNN

Desde hace varios años se busca clasificar en grupos homogéneos las distintas arquitecturas de ordenadores que se van diseñando y produciendo en el tiempo. Esa taxonomía debería comprender también las máquinas paralelas. Existen muchas clasificaciones debidas a distintos especialistas como Hockney, Kuck, Gajski, Treleaven y otros. Pero la que ha llegado hasta nuestros días, anticipándose incluso a arquitecturas no existentes en aquel entonces, y utilizada en la docencia de arquitectura de computadores, es la aparecida al final de los años 60. Se debe a M. J. Flynn y la introdujo en 1966. Está basada en el número de flujos o canales para mover las instrucciones desde la memoria al procesador, *único* o *múltiple* y en el número de flujos existentes para trasladar los datos desde memoria al procesador, también *único* o *múltiple*.

SISD. Flynn considera una máquina secuencial von Neumann aquella que utiliza un *único* flujo o canal para las instrucciones y que trabaja sobre un flujo de datos también *único*. El término inglés para designar este tipo de máquina es **SISD**, *Single Instruction stream Single Data stream*. En la Figura 1 ya presentamos un esquema simplificado de esta máquina secuencial. De forma muy básica, algunas de sus componentes a destacar, son:

- El **procesador**, también conocido con el nombre de *Unidad Central de Proceso, UCP* o *CPU*, en inglés, dentro del cual se encuentra la *Unidad de Control* que se encarga de leer las instrucciones de un programa y ordena a la unidad de tratamiento que efectúe las operaciones respectivas sobre los datos. Profundizando un poco más en la *CPU* que es el corazón del computador y por lo tanto responsable de todos los cálculos así como de controlar y supervisar el resto de componentes de la máquina, debemos considerar:
  - La *Unidad Aritmético-Lógica*, en inglés *ALU*, *Arithmetic Logic Unit*, que ejecuta los cálculos propiamente dichos, tales como una suma, o las comparaciones.
  - La *Unidad de Coma Flotante*, en inglés *FPU*, *Floating Point Unit*, encargada de ejecutar operaciones con números reales.
  - *Unidad de Carga/Almacenamiento*, en inglés *LSU*, *Load/Store Unit*, que efectúa la carga de los datos en el procesador y del almacenamiento de datos y resultados en la memoria principal.
  - *Registros*. Tipo de memoria rápida empleada para almacenar resultados intermedios que puedan obtenerse en un cálculo.
  - *Contador de Programa*, en inglés *PC*, *Program Counter*, registro que contiene la dirección de memoria donde se encuentra la instrucción que se está ejecutando en cada instante.
- La **memoria principal** o memoria central donde se almacenan datos y programas. Además, el chip de la CPU contiene también una memoria más rápida que la principal que se denomina *memoria caché*.

La CPU trabaja en etapas controladas por un **reloj**: en cada etapa o *ciclo de reloj*, la CPU ejecuta una operación.

Lo que hemos dicho y aplicado a una máquina secuencial se extiende a cualquier sistema paralelo: una

máquina paralela está compuesta básicamente por un conjunto de CPU secuenciales.

Las tres categorías siguientes recogen, de alguna forma, supercomputadores o sistemas paralelos, aunque realmente paralelos solo son las dos últimas clases que vamos a describir. Daremos una sucinta descripción de las tres.

SIMD. La arquitectura **SIMD**, *Single Instruction stream Multiple Data stream*, consta de un conjunto de unidades de tratamiento o proceso regidas por una sola unidad de control que se encarga de analizar y decodificar las instrucciones. Se muestra un esquema en la Figura 8. En esta arquitectura, el flujo de instrucciones es único mientras que el de datos es múltiple.

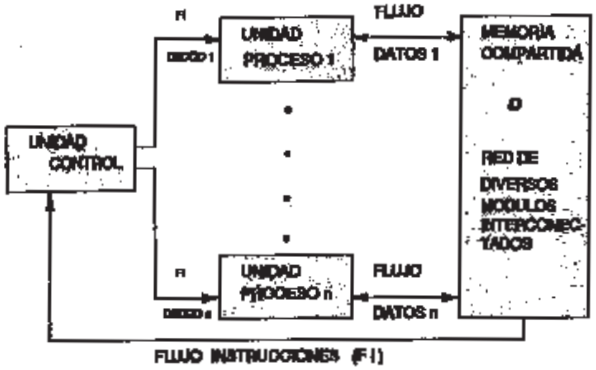

Figura 8. Arquitectura SIMD

Todas las unidades de tratamiento efectúan, en un momento determinado, la misma operación pero cada una utiliza datos distintos. Es como si todas las operaciones paralelas se ejecutaran síncronamente bajo la supervisión de la unidad única de control central, dentro de la cual se encuentra el reloj del sistema.

Algunas veces, una máquina SIMD puede disponer de más de una unidad de control. En ese caso decimos que se trata de un sistema Múltiple SIMD o *MSIMD*.

Supongamos que es necesario efectuar la suma de un conjunto de datos a_i, por ejemplo mil datos, con otro conjunto b_i, de la misma cantidad de datos, y llevar los resultados de cada suma a otro conjunto de mil elementos, $Suma_i$. El código de programa sería algo parecido a:

Hacer desde i = 1 hasta i = 1000
$Suma_i \leftarrow a_i + b_i$
$Suma \leftarrow + Suma_i$
Fin

La sentencia $Suma \leftarrow + Suma_i$, almacena en la variable ***Suma*** las 1000 sumas parciales.

Entonces, si suponemos que nuestra máquina tuviera cuatro unidades de tratamiento, podríamos hacer que cada una de ellas se encargara de efectuar solo 250 sumas, trabajando todas ellas de forma simultánea, es decir en paralelo con las tres restantes. Con ello, el rendimiento obtenido por el sistema sería máximo. Por fin, sería necesario acumular los resultados obtenidos por cada uno de los cuatro procesadores.

Desde el punto de vista del programador, las cosas ocurren como si las operaciones se ejecutaran en una máquina secuencial von Neumann clásica, con la diferencia de que, en un momento determinado, el sistema ejecuta varias operaciones, aunque realmente es la misma, en lugar de una sola: las máquinas SIMD aparecen, pues, como secuenciales.

Los datos tratados por cada elemento de cálculo o unidad de tratamiento se pueden encontrar bien en un espacio de memoria global común a todas ellas o en espacios de memoria propios y locales a cada unidad de tratamiento.

En el pasado, este tipo de sistema recibió el nombre de *matriz de procesadores*, en inglés *array processor*.

MISD. Se trata de una arquitectura *Múltiple Instruction stream Single Data stream*, varios flujos o canales para circular las instrucciones, pero solo un canal para circulación de los datos. Este tipo de computadores realmente nunca se ha construido aunque su estructura responde, a veces, a las necesidades de ciertas aplicaciones como *reconocimiento de formas* y algún tipo de *bases de datos*. Una de las razones es que todos los problemas que pudieran resolverse en sistemas MISD, también se pueden resolver en los MIMD que describiremos más adelante. La Figura 9 muestra un esquema de su diseño básico. Se trataría

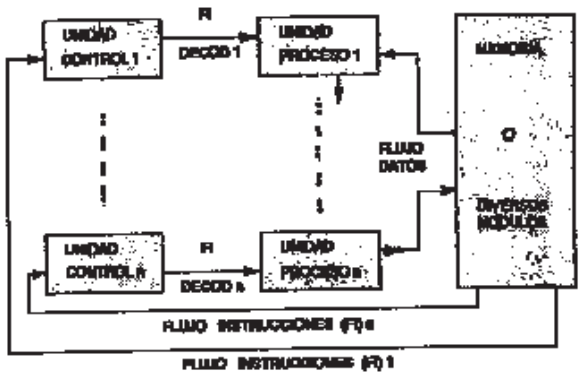

Figura 9. Arquitectura MISD

de una serie de unidades de tratamiento a lo largo de las cuales circularía, por un único canal, un cierto dato el cual estaría sometido a instrucciones diferentes, una por cada unidad de tratamiento por la que el dato pase, instrucción que la unidad de control respectiva se encargaría de enviar a cada una de esas unidades.

En cierto modo se trataría de una arquitectura que trabaja de forma segmentada o en *pipelinning*, por lo que algunos diseñadores quieren ver en esta arquitectura la de una máquina *vectorial* por su trabajo en cadena al existir una sola entrada de datos y varias unidades de cálculo.

MIMD. Arquitectura *Múltiple Instruction stream Multiple Data stream*, es decir, varios canales para las instrucciones y varios canales para los datos. En este sistema se repite varias veces el elemento de proceso, es decir, posee varios procesadores. Ver Figura 10.

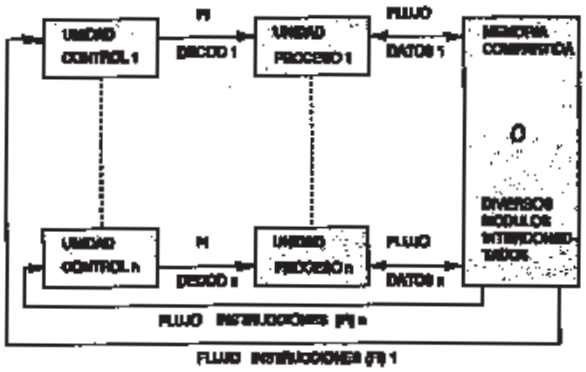

Figura 10. Arquitectura MIMD

Esos procesadores ejecutan tareas o procesos independientes. Si colaboran en la realización de un solo *job*, nos encontraremos con un sistema paralelo multiprocesador. Este tipo de máquina es, con mucho, la más importante desde el punto de vista de arquitectura paralela.

Los procesos que se ejecutan en procesadores distintos, se comportan como si se tratara de cálculos independientes que se realizan de modo asíncrono. No existirían problemas de *dependencias* mientras los procesos trabajen con datos independientes entre sí. Lo que ocurre en la mayoría de los casos es que los procesos cooperan comunicándose vía datos comunes. Entonces, es frecuente que la gestión correcta del acceso concurrente a un dato común origine problemas de sincronización, de *dependencias* y *cuellos de botella*, que son difíciles de resolver.

		Flujo de datos	
		único	múltiple
Flujo de instrucciones	único	SISD (Von Neumann)	SIMD (Matriz procesadores)
	múltiple	MISD (Pipeline)	MIMD (Multiprocesador)

Figura 11. Cuadro resumen clasificación de Flynn

La interacción entre los procesadores para resolver un único problema puede servir para subclasificar estos sistemas en dos categorías: si el grado de interacción o comunicación es bajo y poco frecuente, se dice que el sistema está **débilmente acoplado**, en inglés *loosely couple*; si el grado de interacción es alto, **fuertemente acoplado**, en inglés *tightly couple*.

UN CRITERIO MÁS: LA ORGANIZACIÓN DE LA MEMORIA

Las tres últimas familias de sistemas multiprocesador no permiten clasificar de forma satisfactoria las máquinas paralelas. Y ello, por no facilitar ninguna indicación sobre la organización de la memoria principal, lo que es fundamental para la programación y la utilización de sistemas paralelos.

Desde los años 70 se produce un avance progresivo en la tecnología de la memoria que ha permitido la fabricación de memorias más rápidas y de mayor tamaño.

Vamos, entonces, a proponer una clasificación menos empleada que la de Flynn pero más conforme con la realidad del uso y del mercado. Bajo ese criterio de cómo se comporta y gestiona la memoria, y como paso intermedio para plantear la nueva clasificación, pueden distinguirse dos tipos de máquinas:

- **Máquinas de memoria compartida - SM**
- **Máquinas de memoria distribuida – DM**

Las **máquinas de memoria compartida *SM***, en inglés *Shared Memory*, permiten a todos los procesadores del sistema paralelo acceder a la memoria completa disponible.

La Figura 12 representa un esquema de tales sistemas con varios procesadores y una única memoria principal suficientemente amplia.

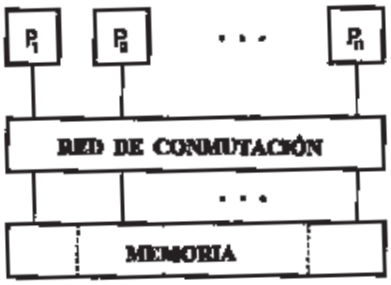

Figura 12. Estructura básica de un sistema paralelo con memoria compartida

En un principio, el uso de este tipo de memoria planteaba problemas físicos al producirse cuellos de botella en el acceso simultáneo, por parte de dos o más procesadores, a una misma dirección o dato contenido en la misma. Evidentemente, fue necesario el arbitraje de las demandas y su secuenciación, aunque con ello se produzcan *contenciones*, es decir, la espera de un procesador por los datos que necesita hasta que la memoria quede liberada por el procesador que la está utilizando en cada instante. La mejor solución para limitar estos efectos ha sido dividir la memoria común y total en *bancos de memoria***,** físicamente distintos pero contiguos, y gestionarlos mediante una técnica computacional conocida con el nombre de *entrelazado*, en inglés *interleaving*. Se consigue así un *espacio de direcciones global*, *global address space*. La comunicación entre procesadores se lleva a cabo mediante esa memoria común y, por lo tanto, no es explícita.

Este tipo de organización de la memoria implica una programación rigurosa por parte del usuario a fin de evitar los conflictos de acceso a una misma dirección por dos procesadores distintos, como ya hemos dicho anteriormente. En todo caso, son máquinas más complicadas de programar que las de memoria distribuida que vamos a analizar un poco más adelante.

En una máquina paralela de memoria compartida, los datos de un programa están almacenados en el mismo recurso: la memoria común. Así, un proceso *p* situado en un procesador *P* tendrá la posibilidad de acceder a los datos de otro proceso *q* almacenado y en ejecución en otro procesador *Q*. Lo cual resulta de máxima utilidad al permitir a los procesos intercambiar informaciones a través de una técnica de programación conocida con el nombre de *variables compartidas*. Cada procesador puede acceder con la misma rapidez a cualquier posición de memoria.

Otro elemento constitutivo de este tipo de máquinas es la **red de interconexión**. Se trata de una vía electrónica de comunicación, técnicamente es una red de conmutación, para conectar todos los procesadores con todos los bancos de la memoria. Cualquier escritura o almacenamiento en la memoria, ordenada por uno de los procesadores, puede considerarse como una comunicación implícita con otro procesador del sistema paralelo que, en un instante posterior, accederá a esa misma posición de la memoria.

En los **sistemas de memoria distribuida *DM***, en inglés *Distributed Memory*, que aparecen también en las arquitecturas paralelas, la memoria ya no está compartida por todos los procesadores, no es única, sino distribuida entre ellos, con lo que cada elemento de proceso dispone de su propia memoria para datos y programas. Véase un esquema en la Figura 13.

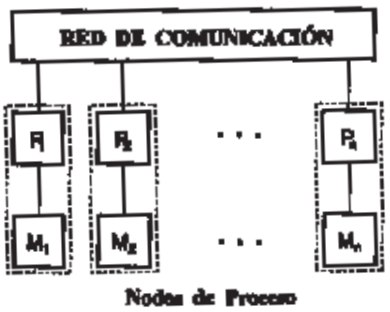

Figura 13. Estructura básica de un sistema paralelo con memoria distribuida

Cada procesador del sistema tiene su propia memoria local a la que puede acceder directamente. Un procesador cualquiera P_i puede acceder a datos almacenados en su memoria local M_i; pero también es posible, en principio, su acceso a la memoria local M_j de otro procesador P_j. En este último caso, el **nodo** *j* que consta del procesador P_j y su memoria local M_j, enviará los datos pedidos al nodo *i* para su proceso, a través de la **red de comunicación** que interconecta entre sí todos los nodos de proceso.

La mayoría de los supercomputadores paralelos actuales pertenecen a este tipo. Su programación, lejos de ser algo trivial, no entraña, sin embargo, las dificultades que se habían mencionado en los sistemas con memoria compartida porque actualmente se dispone de un software de sistemas bastante adecuado y eficiente. El modelo de programación paralela empleado, se conoce con el nombre de *modelo de paso de mensajes*, en inglés *message passing model* y, como veremos más adelante, existen productos software como PVM, MPI, entre otros, con los que resulta relativamente sencillo para el programador efectuar ese correo de mensajes.

El papel de la red de interconexión que figura en los dos últimos gráficos, presenta, como puede observarse, ciertas diferencias según se trate de una u otra familia. Para no entrar en detalles que sobrepasarían los límites técnicos que nos hemos propuesto, diremos simplemente que, en un ordenador *SM* la red conecta el conjunto de procesadores al conjunto de bancos de memoria; en un ordenador *DM*, la red conecta unos procesadores con otros.

Llegamos así a una clasificación de arquitecturas unificada que es más realista y práctica que la de Flynn. Combina las clases SIMD y MIMD con la organización de la memoria que hemos descrito, es decir, con las clases SM y DM. Ver Figura 14.

El control de la ejecución secuencial de las instrucciones por medio de un secuenciador, puede estar centralizado o distribuido. Si está centralizado, será un control único, *SI*; si está distribuido, se tratará de un control múltiple, *MI* . Los datos siempre son varios o múltiples, *MD*, y pueden estar almacenados en una memoria compartida, *SM*, o en memorias distribuidas, *DM*.

En el gráfico, los elementos de cálculo indicados con ***P***, disponen de su propio secuenciador; los indicados por ***PE***, operan bajo control de un secuenciador de instrucciones externo.

De esta forma, y tal como se ilustra en la Figura 14, llegamos a las cuatro clases que se reconocen por sus respectivas siglas: SIMD-SM, SIMD-DM, MIMD-SM, MIMD-DM.

¿ES IMPORTANTE LA ARQUITECTURA *RISC*?

Un concepto que se utiliza mucho al hablar de computadores de alto rendimiento es el relacionado con los *procesadores RISC*, del ingés *Reduced Instruction Set Computer* que se refiere a un computador con un procesador de relativamente muy pocas instrucciones, normalmente menos de 100. Del otro lado están los procesadores comunes con un juego de instrucciones muy amplio, de entre 200 y 300 junto con una variedad de modos distintos de direccionamiento, y que se conocen con el nombre de *procesadores CISC*, del inglés *Complex Instruction Set Computers*.

La idea detrás de RISC es la de disponer de un sistema capaz de ejecutar casi todas las instrucciones de

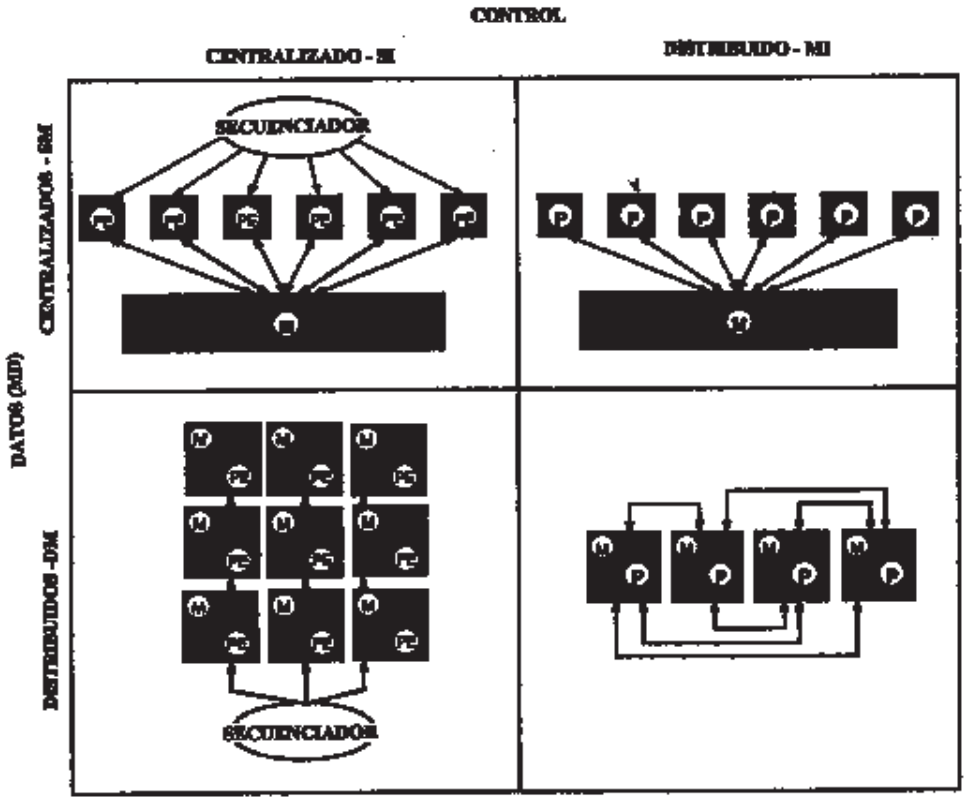

Figura 14. Cuatro clases de arquitecturas paralelas

su juego en un solo ciclo de reloj cada una y las que precisen de más de un ciclo, su ejecución se haga de modo segmentado o en *pipelinning*, todo lo cual contribuye, una vez más, a conseguir altos rendimientos que, ya sabemos, es la premisa básica para la supercomputación.

La conclusión es: estas ideas prueban que con este tipo de arquitectura RISC se consiguen altos rendimientos no solo en ordenadores pequeños, como son las estaciones de trabajo, *workstations*, sino también en la mayor parte de los sistemas masivamente paralelos que actualmente están diseñados y producidos con ese tipo de procesador.

PARA DIALOGAR.... NECESIDAD DE UN LENGUAJE

Los rápidos avances en la tecnología del hardware han hecho posible sistemas de computación que se alejan mucho del modelo de máquina secuencial de Von Neumann encontrándonos actualmente con sistemas paralelos con una relación casi óptima precio-efectividad. Como ya hemos dicho, esas máquinas las tenemos en el mercado.

Existen, sin embargo, grandes barreras para la utilización del paralelismo en cualquier hardware debido a la ausencia de herramientas eficientes de programación como compiladores y depuradores para el desarrollo de aplicaciones con estas máquinas de alto rendimiento.

Teniendo en cuenta esta dificultad, surgen, en consonancia, cuestiones a plantearse:

- Cómo extraer el paralelismo a partir de la descripción de un problema que era fácilmente computable en una máquina secuencial.
- Dada una determinada máquina paralela, cómo es posible descomponer un problema para obtener el máximo rendimiento en su ejecución.
- Cómo puede medirse el rendimiento y la eficiencia de esa ejecución.

Algunas respuestas a estas preguntas las daremos ahora y un poco más adelante.

Básicamente, y desde el punto de vista de los compiladores, nos encontramos con tres caminos para programar supercomputadores:

- Hacer uso de un lenguaje secuencial ya existente y conseguir que todo el trabajo de paralelización lo haga el compilador. Es necesario, para ello, que dicho compilador tenga un cierto nivel de *inteligencia*. Es lo que se conoce con el nombre de *aproximación implícita*.

- Tomar un lenguaje secuencial ya existente y agregarle nuevas construcciones sintácticas para expresar el paralelismo. Es la *aproximación explícita*.

- Desarrollar un nuevo lenguaje de programación paralela. Ello supone tener que recodificar todo el software de aplicación, lo que entraña altos costes y riesgos.

Una vez escogido uno de estos tres modelos, queda la cuestión de cómo seleccionar un lenguaje de programación entre los disponibles en el mercado. Esto entraña una dificultad adicional ya que muchos lenguajes son mixtos, en el sentido de contener algo de los tres modelos enunciados más arriba.

Enumeraremos alguno de estos lenguajes sin entrar en detalles sobre los mismos.

- *Parallel Fortran* del grupo PCF, *Parallel Computing Forum*, lenguaje estandarizado pero con muy poco uso generalizado.

- *Fortran 90 ó 95*. Es una extensión de Fortran 77 que permite el tratamiento paralelo de vectores, matrices y operaciones entre ellos.

- HPF, *High Performance Fortran*, trabaja sobre Fortran 90 ó 95 y está diseñado para su utilización sobre sistemas paralelos de memoria distribuida.

- Ganando popularidad se encuentran derivados de los lenguajes C y su versión orientada a objetos, C++. Aunque ambos son de gran complejidad y de semántica complicada para aplicarlos a la programación paralela.

- *OpenMP*, conjunto de directivas para el compilador, rutinas de biblioteca y variables de entorno que pueden emplearse tanto en Fortran como en C/C++ para especificar paralelismo en sistemas de memoria compartida y ello, dentro de bucles de programación repetitivos.

EL SOPORTE QUE NO PUEDE FALTAR: LA CLAVE DE LA BÓVEDA

Una de las razones para la dificultad de uso de los sistemas MIMD del pasado era la ausencia de software adecuado. Tales máquinas estaban reservadas solo a algunos iniciados en el paralelismo, mientras que los que realmente necesitaban gran potencia de cálculo no podían utilizarlas de forma razonable. Además, la falta de portabilidad y los desmesurados tiempos de desarrollo dejaban al paralelismo solo a la puerta del mundo industrial.

De todo el software matemático estándar, el de uso más frecuente, según revelan numerosos estudios, quizás sea el constituido por rutinas de álgebra lineal. La solución de un problema de álgebra lineal suele formar parte de problemas computacionales más complejos. Como consecuencia, se ha invertido mucho y se han realizado grandes esfuerzos para optimizar las operaciones en este álgebra con el fin de que pudieran ser ejecutadas en diferentes arquitecturas.

Portabilidad y eficacia son las dos palabras clave de la algorítmica paralela actual. Ello unido a la optimización que se consigue al producir el software anejo, hacen de las bibliotecas de rutinas una herramienta imprescindible para el programador.

Con una colección de rutinas estándar pueden resolverse en máquinas paralelas, entre otros, problemas técnicos y científicos, como:

- Solución de sistemas de ecuaciones lineales.
- Problemas de mínimos cuadrados.
- Problemas de valores y vectores propios.
- Álgebra de matrices.

En las tres últimas décadas se han desarrollado rutinas de software de alta calidad con el objetivo de ser ejecutadas en supercomputadores. Citaremos solo algunos nombres.

- EISPACK, 1972, escrito en Fortran 66 y orientado a la resolución de problemas generales sobre valores/vectores propios.

- BLAS, *Basic Linear Algebra Subroutines*, 1978, con rutinas para cálculos de álgebra lineal básica. Fuero definidas por C. Lawson, R. Hanson, D. Kincaid y F. Krogh.

- LINPACK, 1979, para problemas de sistemas de ecuaciones y de mínimos cuadrados.

- BLAS 2, BLAS 3, definidas por J. Dongarra y LAPACK, *Linear Algebra Package*, 1985 – 1992, son nuevas versiones de EISPACK, BLAS y LINPACK, para adaptar esas versiones primitivas a las nuevas arquitecturas vectoriales y multiproceso que emergían en los años 80 y conseguir así una biblioteca de alto nivel. LAPACK proviene de la fusión de las EISPACK y LINPACK.

- ScaLAPACK, *Scalable LAPACK*, es la biblioteca LAPACK para sistemas paralelos de memoria distribuida.

- PBLAS, *Parallel BLAS*, colección de rutinas paralelas producidas por un equipo dirigido por J. Dongarra para simplificar el desarrollo de las versiones, también paralelas, de LAPACK y ScaLAPACK.

- NAG es el nombre de una compañía de desarrollo de software que produce una biblioteca de programas para la solución de problemas numéricos constantemente actualizada y ampliada. Rutinas semejantes americanas se denominan IMSL.

La mayoría de estos paquetes de rutinas están escritos en Fortran, estándares 77, 90 y 95. Actualmente también se dispone de versiones en lenguaje C, como ocurre con NAG e IMSL.

Uno de los aspectos más importantes de los altos rendimientos de los sistemas paralelos de memoria distribuida, hemos dicho que era su capacidad para comunicarse sus procesadores entre sí. Las bibliotecas de comunicación constituyen la base de la programación de tales máquinas.

En este campo los objetivos son múltiples y, a veces, antieconómicos. Por eso el desarrollo se fija sólo teniendo en cuenta o bien alcanzar buenos rendimientos o conseguir una fácil utilización. El ideal es lograr ambos.

Vamos a citar solo tres de estas bibliotecas: BLACS, PVM, MPI.

- BLACS, *Basic Linear Algebra Communication Subroutines*. Dedicadas a las operaciones de comunicación empleadas para la paralelización de las rutinas BLAS o LAPACK. También se utili-

zan para el movimiento de matrices algebraicas entre procesadores.

- PVM, *Parallel Virtual Machine*. Una tendencia actual es poder comunicar de manera económica varias estaciones de trabajo, por ejemplo varios ordenadores personales, para simular así una máquina paralela o para formar una serie de máquinas distribuidas heterogéneas, es decir, de distintas marcas o modelos. Gracias a PVM pueden conseguirse tales redes. Ha sido desarrollado en Oak Ridge National Laboratory por un equipo de la Universidad de Tennessee dirigido por J. Dongarra y otro de la Universidad de Emmory bajo la dirección de V. S. Sunderam. PVM está formado por un conjunto de *demonios* UNIX y por una biblioteca de comunicación que permite la creación y paso de mensajes en un entorno con varios sistemas UNIX. Puede utilizarse desde C y desde Fortran.

- MPI, *Message Passing Interface*. Esta biblioteca es la especificación de un estándar para el intercambio de mensajes entre máquinas de memoria distribuida. Su objetivo: tener acceso a una misma interface, sencilla de utilizar, sobre todas las máquinas de un mismo tipo. Con estas rutinas se ha logrado cubrir una amplia variedad de aplicaciones dentro de la computación gracias a investigadores y desarrolladores de software que han trabajado con ese propósito.

Para finalizar con este tema relativo a las ayudas de base para la programación paralela, solo unas líneas más para referirnos a las rutinas encargadas de medir el rendimiento de un sistema de tales características: herramientas para el análisis de rendimientos mediante visualización gráfica en la pantalla.

PARAGRAPH es uno de estos sistemas con el que puede visualizarse el comportamiento y los rendimientos de los programas paralelos sobre arquitecturas de paso de mensajes. Es posible hacer el seguimiento de la ejecución de las tareas deseadas de un programa total, afectándolas de diferentes colores lo que permitirán así apreciar el tiempo empleado en cada una de ellas. Para ello, se dispone de gráficos como:

- *Spacetime*, vista del conjunto de la ejecución con todos los procesadores disponibles del sistema, en las ordenadas y, como abscisas, el tiempo.

- *Critical Path*, Camino Crítico, semejante a Spacetime pero el camino crítico del programa se representa en color rojo.

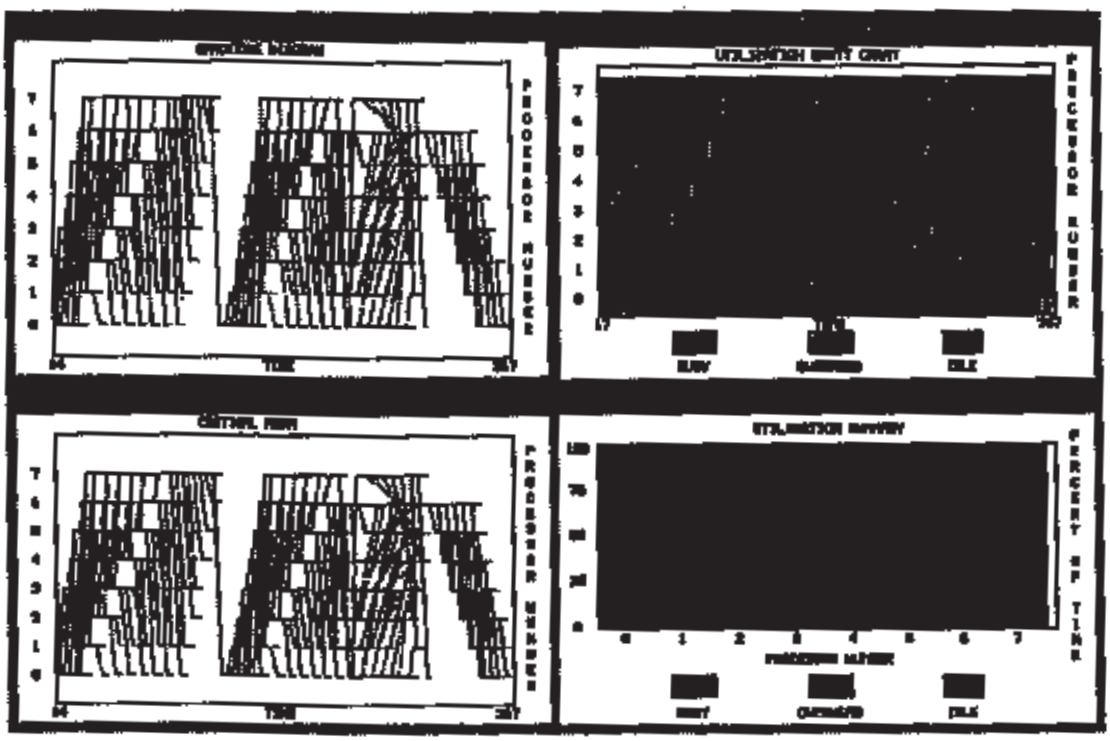

Figura 15. PARAGRAPH: Spacetime, Camino Crítico, Gráfico de Gantt y Summary

- *Gantt*, presenta la actividad, ocupado, en sobrecarga, en espera, de los procesadores a lo largo del tiempo.
- *Summary*, como su nombre indica, es un resumen de la actividad de los procesadores durante la ejecución del programa completo.

A MODO DE RESUMEN Y CONCLUSIÓN

La arquitectura de los ordenadores secuenciales ha mejorado mucho desde las primitivas máquinas de los años 40 y 50. Existen, sin embargo, limitaciones físicas, tales como la velocidad de la luz, que constituyen una barrera natural para hacer del procesamiento secuencial algo ilimitado. El paralelismo ofrece hoy la forma de incrementar la potencia computacional más allá de esas barreras.

Procesamiento paralelo es la solución de un único problema mediante el uso de más de un elemento de proceso y de técnicas de segmentación o de empleo repetido de procesadores.

Conceptualmente el procesamiento paralelo es algo muy sencillo: subdivisión de un problema en un cierto número de subproblemas. Lo que resulta más complicado es la implementación de ese concepto en un sistema multiprocesador.

Para resolver el problema es necesaria la cooperación entre los elementos de proceso o los procesadores.

Con el paralelismo y la supercomputación es posible resolver, no solo los problemas que se nos presentan hoy día en fracciones del tiempo empleado hace algunos años, sino también otros más complejos e insolubles en el pasado.

Todavía existen problemas que precisan de más potencia computacional que la disponible en el mercado actual para que puedan resolverse en tiempos razonables. Pero es el futuro que ya está llamando a nuestras puertas.

La flexibilidad y la potencial escalabilidad ofrecida por las arquitecturas MIMD de memoria distribuida hacen de estas máquinas las más convenientes para el procesamiento paralelo práctico.

Uno de los principales objetivos de la investigación y desarrollo en el procesamiento paralelo tiene que ser, necesariamente, el de reducir el costo de la escritura y de la ejecución de los programas paralelos, y en particular la de aquellos que emplean sistemas multiprocesador de memoria compartida. Este objetivo será, incluso, más difícil de alcanzar a medida que las plataformas de computación se hagan más complejas en el futuro.

*Leía no hace mucho en **Beyon Calculation, The nex fifty years of Computing**, libro de divulgación editado hace muy pocos años por P.J. Denning, Virginia, y R. M. Metcalf, Boston, lo siguiente: "Los computadores de alto rendimiento están llegando, y lo seguirán haciendo, pero tienen un oscuro futuro: la mayor parte de ellos poseen estructuras diferentes por lo que requieren también el correspondiente software diferente. ¿Será posible escribir un software razonablemente eficiente para generar los compiladores correspondientes? o ¿Estaremos avocados a realizar una ingente cantidad de trabajo específico para cada familia de computadores paralelos? ". Las frases son de R. W. Hamming.*

*La **arquitectura paralela** continúa siendo un área activa y excitante de la investigación, dentro de las tecnologías del procesamiento de la información.*

La perfección se consigue
no cuando ya no queda nada por añadir,
sino cuando no queda nada por suprimir.

Antonio de Saint Exupéry