ORDENADORES VECTORIALES

Un ordenador vectorial es una máquina diseñada específicamente para realizar de forma eficiente operaciones en las que se ven involucrados elementos de matrices, denominados vectores. Estos ordenadores resultan especialmente útiles para ser utilizados en el cálculo científico de alto rendimiento (high performance computing), donde las operaciones con vectores y con matrices son ampliamente utilizadas.

Arquitectura general

Vectores y aritmética vectorial

Para mostrar los conceptos que se encuentran detrás de un procesador vectorial, vamos a presentar en primer lugar una breve muestra de cómo programar operaciones vectoriales sobre códigos FORTRAN.

Un vector v, es una lista de elementos de la forma:

$$v = (v1, v2, v3,..., vn)T$$

La longitud del vector se define como el número de elementos en el vector, de forma que la longitud del vector v que acabamos de presentar es n. Cuando queremos representar un vector en un programa, se declara el vector como una matriz de una única dimensión. En FORTRAN, declararíamos el vector v mediante la expresión:

```
DIMENSION V (N)
```

En donde N es una variable entera que almacena el valor de la longitud del vector.

Dos vectores se pueden sumar simplemente sumando cada una de sus componentes, es decir:

$$s = x + y = (x1+ y1, x2+ y2, . . . , xn+yn)$$

En FORTRAN, la suma de vectores se puede realizar utilizando el siguiente código:

```
DO I=1, N

S (I) = X(I) + Y (I)

ENDDO
```

Donde s es el vector en el que se almacena la suma final, y S, X, e Y han sido declarados como matrices de dimensión N.

Elementos de la arquitectura vectorial

Un ordenador vectorial contiene un conjunto de unidades aritméticas especiales denominadas pipelines. Estas pipelines superponen la ejecución de las diferentes partes de una operación aritmética sobre los elementos del vector, produciendo una ejecución más eficiente de la operación aritmética

que se está realizando. En muchos aspectos, una pipeline es similar a una cadena de montaje de una fábrica de coches, en la que los distintos pasos de la fase de montaje de un automóvil, por ejemplo, se realizan en distintas etapas de la cadena.

Consideremos, por ejemplo, los pasos o etapas necesarias para realizar una suma en punto flotante en una máquina con hardware que emplee aritmética IEEE. Los pasos para realizar la operación s = x + y son:

1. Se comparan los exponentes de los dos números en punto flotante que se quieren sumar para encontrar cuál es el número de menor magnitud.

2. El punto decimal del número con menor magnitud se desplaza de forma que los exponentes de los dos números coincidan.

3. Se suman los dos números.

4. Se normaliza el resultado de la suma.

5. Se realizan los chequeos necesarios para comprobar si se ha producido algún tipo de excepción en punto flotante durante la suma, como un overflow.

6. Se realiza el redondeo.

La tabla 1 muestra los pasos necesarios para realizar esta suma. Los números que se van a sumar son el x = 11234.00 y el y = -567.8. A diferencia de la representación habitual, estos números se almacenan en notación decimal con una mantisa de cuatro dígitos.

Ahora consideremos esta suma escalar realizada sobre todos los elementos de un par de vectores de longitud n. Se deben realizar cada una de las seis etapas para cada par de elementos de los vectores. Si en cada fase de la ejecución son necesarias t unidades de tiempo, entonces cada suma necesita 6t unidades de tiempo (sin contar el tiempo necesario para suministrar y decodificar la instrucción en sí o para traer los dos operandos hasta la unidad aritmética). Como resultado, el número de unidades de tiempo necesarias para sumar todos los elementos de los dos vectores en serie sería de ts= 6nt. Las distintas fases de la ejecución en función del tiempo se muestran en la tabla 2.

Paso	A	B	C	D	E	F
x	0.1234E4	0.12340E4				
y	-0.5678E3	-0.05678E4				
z			0.066620E4	0.066620E3	0.066620E3	0.066620E3

Táboa 1. Un exemplo para mostra-los pasos que se seguen ó realiza-la suma de dous números en punto flotante: s= x + y

La pipeline aritmética

Vamos a mostrar ahora cómo segmentar la operación anterior, esto es, hacer que cada una de las seis fases necesarias para sumar cada par de elementos se realice en cada fase de la cadena (pipeline). En cada paso de esta cadena existe una unidad aritmética separada diseñada para realizar la operación que se desea en cada fase. Una vez que la fase A ha sido completada para el primer par de elementos, estos elementos pueden ser desplazados hasta la siguiente fase (B) mientras que el segundo par de elementos se puede mover hacia la primera fase (A). De nuevo, cada fase requiere t unidades de tiempo. Por tanto, el flujo a través del pipeline puede ser representado como se muestra en la figura1, en donde cada una de las fases de la suma segmentada se ejecutan en función del tiempo tal y como se indica en la tabla3.

Paso	t	2t	3t	4t	5t	6t	7t	8t
A	x1 + y1						x2 + y2	
B		x1 + y1						x2 + y2
C			x1 + y1					
D				x1 + y1				
E					x1 + y1			
F						x1 + y1		

Táboa 2. Suma escalar en punto flotante dos elementos dun vector.

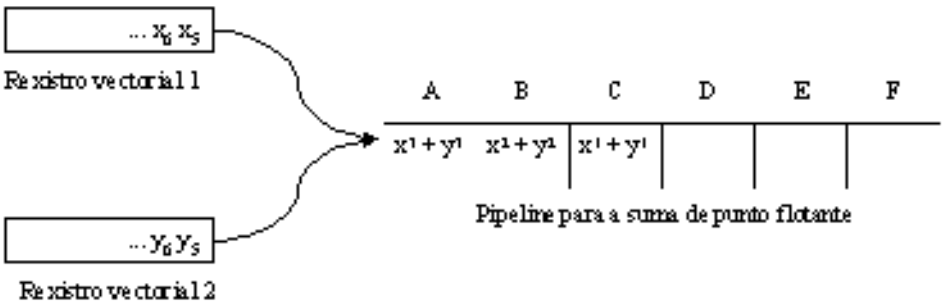

Fig. 1. *Pipeline* utilizada para a suma en punto flotante dos elementos dun vector.

Como se puede ver, todavía son necesarias 6t unidades de tiempo para completar la suma del primer par de elementos, pero también se puede comprobar cómo el siguiente par de elementos está listo en tan sólo t unidades de tiempo después. Y este patrón continúa para cada uno de los pares siguientes.

Esto significa que el tiempo total, Tp, para realizar la operación de forma segmentada sobre dos vectores de longitud n es:

$$Tp = 6t + (n-1)t = (n + 5)t$$

Las primeras 6t unidades de tiempo son necesarias para llenar el pipeline y para obtener el primer resultado. Después de que se halla calculado el último resultado, xn + yn, el pipeline se encuentra vacío.

Comparando las ecuaciones para Ts y Tp, resulta evidente que (n+5)t < 6nt, para n>1. Por tanto, la versión segmentada de la suma resulta más rápida que la versión secuencial por un factor casi igual al número de fases que existen en el pipeline. Este es un ejemplo que muestra por qué el procesamiento vectorial resulta más eficiente que el procesamiento escalar. Para valores grandes de n, la suma segmentada utilizando este pipeline es casi seis veces más rápida que la suma escalar.

En el ejemplo anterior, hemos asumido que la suma en punto flotante requiere seis etapas y que por tanto utiliza 6t unidades de tiempo. Sin embargo, en determinadas arquitecturas el número de fases necesario para realizar la suma en punto flotante puede ser mayor o menor de seis. Las operaciones necesarias para realizar la multiplicación de dos números en punto flotante también son ligeramente diferentes a las necesarias para realizar la multiplicación, y por tanto, normalmente el número de fases necesario también suele ser distinto. Además, también pueden existir pipelines para realizar operaciones sobre números enteros.

Paso	t	2t	3t	4t	5t	6t	7t	8t
A	x1 + y1	x2 + y2	x3 + y3	x4 + y4	x5 + y5	x6 + y6	x7 + y7	x8 + y8
B		x1 + y1	x2 + y2	x3 + y3	x4 + y4	x5 + y5	x6 + y6	x7 + y7
C			x1 + y1	x2 + y2	x3 + y3	x4 + y4	x5 + y5	x6 + y6
D				x1 + y1	x2 + y2	x3 + y3	x4 + y4	x5 + y5
E					x1 + y1	x2 + y2	x3 + y3	x4 + y4
F						x1 + y1	x2 + y2	x3 + y3

Táboa 3. Suma segmentada dos elementos dun vector en punto flotante.

Registros vectoriales

Algunos ordenadores vectoriales, como el Cray Y-MP, contienen registros vectoriales. Un registro de propósito general o un registro de punto flotante contiene un único valor. Sin embargo, los registros vectoriales contienen en su interior muchos elementos de un vector al mismo tiempo. Por ejemplo, los registros vectoriales del Cray Y-MP contienen 64 elementos, mientras que los del Cray C90 contienen 128 elementos. Los contenidos de estos registros pueden ser enviados a (o recibidos por) una pipeline vectorial a razón de un elemento por paso temporal..

Registros escalares

Los registros escalares se comportan de la misma forma que los registros de punto flotante, conteniendo un único valor. Sin embargo, estos registros se configuran de tal forma que pueden ser utilizados por una pipeline vectorial. En este caso, el valor del registro se lee una vez cada t unidades de tiempo y se introduce en el pipeline, de la misma forma que cada elemento de un vector se

introduce en cada paso en la pipeline vectorial. Esto permite que los elementos de un vector puedan realizar operaciones con elementos escalares. Por ejemplo, para calcular el resultado de la operación y = 2.5x, el valor 2.5 se almacena en un registro escalar y se introduce en la pipeline de multiplicación cada t unidades de tiempo para ser multiplicado por cada uno de los elementos de x y así obtener el resultado en el vector y.

Encadenamiento

La figura 1 muestra el diagrama de funcionamiento de un único pipeline. Como indicamos anteriormente, la mayoría de las arquitecturas vectoriales tienen más de un pipeline, y en muchas ocasiones presentan distintos tipos de pipelines.

Algunas arquitecturas vectoriales proporcionan mayor eficiencia al permitir que la salida de un pipeline se pueda encadenar con la entrada de otro pipeline. Esta característica se conoce con el nombre de encadenamiento y elimina la necesidad de almacenar el resultado del primer pipeline antes de enviarlo al segundo pipeline. La figura 2 demuestra la utilización del encadenamiento para el cálculo de una operación vectorial: s = ax + y, donde x e y son vectores y a es una constante escalar.

El encadenamiento puede duplicar el número de operaciones en punto flotante que se realizan en t unidades de tiempo. Una vez que los pipelines de suma y multiplicación se han llenado, se pueden realizar una operación de suma y otra de multiplicación en punto flotante (es decir un total de dos operaciones en punto flotante) cada t unidades de tiempo.

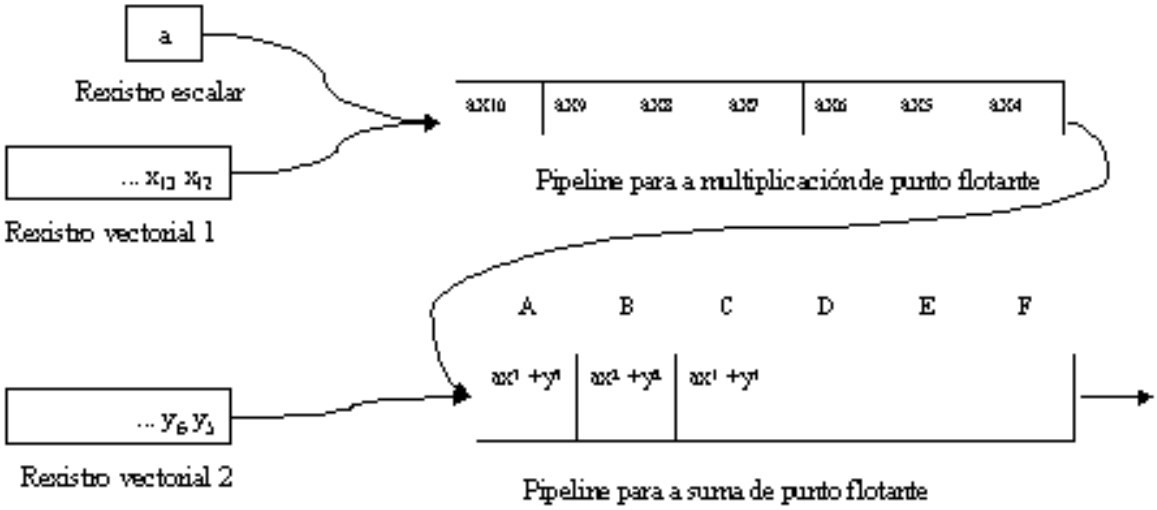

Fig.2.- Encadeamento empregado para realiza-la operación ax + y.

Operaciones de dispersión y agrupamiento

Algunas veces sólo son necesarios algunos de los elementos de un vector para realizar un cálculo. La mayoría de los procesadores vectoriales están equipados para poder recoger los elementos necesarios de un vector (una operación de recogida) y colocarlos juntos en un vector o en un registro vectorial. Si los elementos que se utilizan presentan un patrón de espaciado regular, el espaciado entre los elementos que se van a recoger se denomina desplazamiento o stride. Por ejemplo, si se quieren extraer los elementos:

x1,x5,x9,x13 ,..., x4(n-1)+1

del vector

(x1, x2, x3, x4, x5, x6,...,xn)

para realizar algún tipo de operación vectorial, se dice entonces que el desplazamiento es igual a 4. Una operación de dispersión reformatea el vector resultante para que los elementos se encuentren espaciados correctamente. Las operaciones de dispersión y recogida también se pueden utilizar con datos que no se encuentran regularmente espaciados.

Procesadores vectoriales de registro vectorial

Si un procesador vectorial posee registros vectoriales, los elementos del vector que se van a procesar se cargan desde la memoria directamente en el registro vectorial utilizando una operación de carga vectorial. El vector que se obtiene a partir de una operación vectorial se introduce en un registro vectorial antes de que se pueda almacenar de nuevo en la memoria mediante una operación de almacenamiento vectorial. Esto permite que se pueda utilizar en otra operación sin necesidad de volver a leer el vector, y también permite que el almacenamiento se pueda solapar con otro tipo de operaciones. En este tipo de ordenadores, todas las operaciones aritméticas ó lógicas vectoriales son operaciones registro a registro, es decir, sólo se realizan operaciones vectoriales sobre vectores que ya se encuentran almacenados en los registros vectoriales. Por este motivo, a estos ordenadores se les conoce como procesadores vectoriales de registro vectorial. La figura 3 muestra la arquitectura de registros y pipelines de un ordenador vectorial de registro vectorial. El número de etapas de cada pipeline se muestra entre paréntesis dentro de cada pipeline.

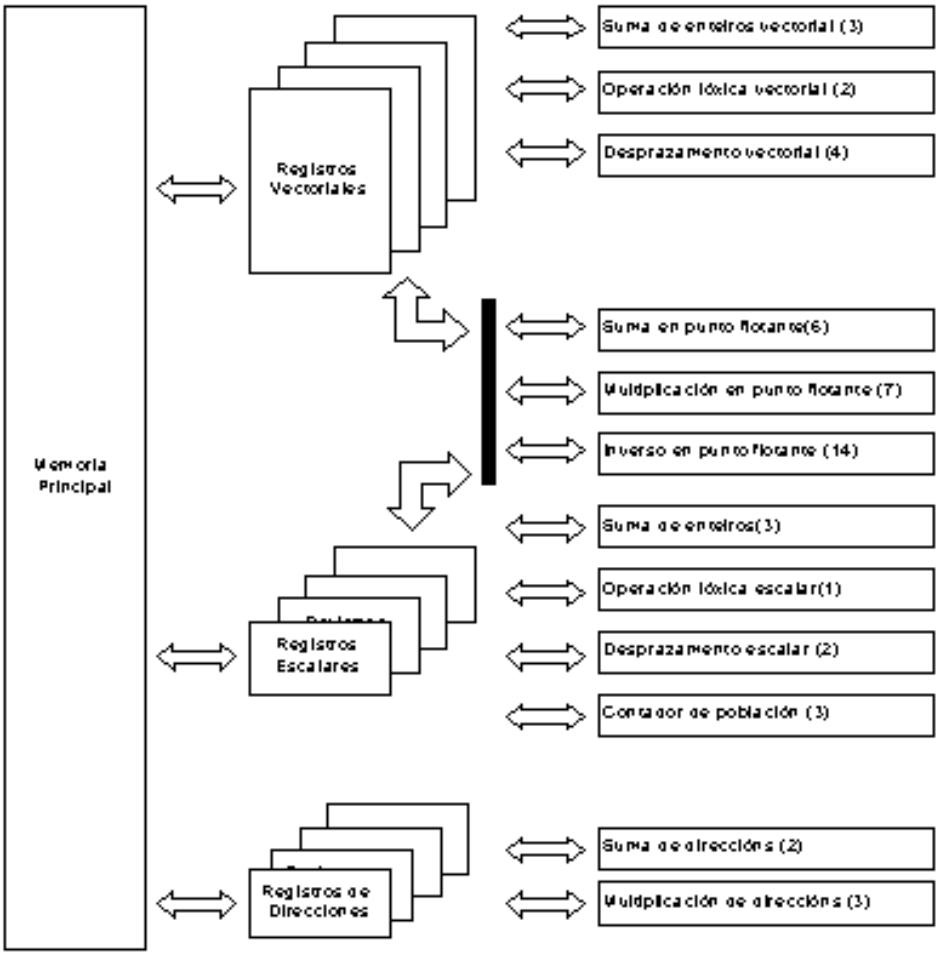

Figura 3. Arquitectura básica dun ordenador Cray-1 cos seus rexistros e *pipelines*. O número que se encontra entre parénteses en cada *pipeline* representa o número de etapas do *pipeline*.

Pro

cesadores vectoriales memoria a memoria

Otro tipo de procesadores vectoriales permite que las operaciones realizadas con vectores se alimenten directamente de datos procedentes de la memoria hasta los pipelines vectoriales y que los resultados se escriban directamente en la memoria. Este tipo de procesadores se conocen con el nombre de procesadores vectoriales memoria a memoria. Dado que los elementos del vector necesitan venir de la memoria en lugar de proceder de un registro, se requiere más tiempo para conseguir que la operación vectorial comience a realizarse. Esto es debido en parte al coste del acceso a la memoria. Un ejemplo de procesador vectorial memoria a memoria era el CDC Cyber 205.
Debido a la capacidad de superponer el acceso a la memoria y la posible reutilización de los vectores ya utilizados, los procesadores vectoriales de registro vectorial suelen ser más eficientes que los procesadores vectoriales memoria a memoria. Sin embargo, a medida que la longitud de los vectores utilizados para un cálculo se incrementa, esta diferencia en el rendimiento entre los dos tipos de arquitecturas tiende a desaparecer. De hecho, los procesadores vectoriales memoria a memoria pueden llegar a ser más eficientes si la longitud de los vectores es lo suficientemente grande. Sin embargo, la experiencia ha demostrado que la longitud de los vectores suele ser mucho más corta de la necesaria para que esta situación llegue a producirse.

Bancos de memoria entrelazados

Para permitir un acceso más rápido a los elementos vectoriales que se encuentran almacenados en la memoria, las memorias de los procesadores vectoriales se suelen dividir en bancos de memoria. Los bancos de memoria entrelazados asocian de forma sucesiva las direcciones de memoria con bancos sucesivos de forma cíclica. De esta forma, la palabra 0 se almacena en el banco 0, la palabra 1 se almacena en el banco 1, ..., la palabra n-1 se almacena en el banco n-1, la palabra n en el banco 0, la palabra n+1 en el banco 1,..., etc., en donde n es el número de bancos de memoria. Como sucede con muchas otras características de arquitectura de ordenadores, n es normalmente una potencia de 2: n= 2k, donde k = 1, 2, 3 ó 4.

Un acceso a memoria (carga o almacenamiento) de un valor de datos en un banco necesita varios ciclos de reloj para llegar a completarse. Cada banco de memoria permite sólo que se lea o almacene un valor de los datos por cada acceso a memoria, mientras que se puede acceder a varios bancos de memoria de forma simultánea. Cuando los elementos de un vector que se almacena en una memoria entrelazada se trasladan al registro vectorial, las lecturas se reparten entre los bancos de memoria, de forma que un elemento vectorial es leído en cada banco por cada ciclo de reloj. Si un acceso a memoria precisa de n ciclos de reloj, entonces n elementos de un vector pueden ser leídos con el mismo coste del que sería necesario para un único acceso a memoria. Este procedimientos es n veces más rápido que el necesario para realizar el mismo número de accesos a memoria sobre un único banco.

Rendimiento de los ordenadores vectoriales

Para las arquitecturas vectoriales más usadas, el valor de t (el tiempo necesario para completar una fase de la cadena ó pipeline) es equivalente a un ciclo de reloj de la máquina. Una vez que el pipeline como el que se muestra en la figura 1 se ha llenado, se genera un resultado para cada t unidades de tiempo, es decir, para cada ciclo de reloj. Esto significa que el hardware es capaz de realizar una operación de punto flotante por cada ciclo de reloj.

Si k representa el número de t unidades de tiempo que la misma operación requeriría sobre una máquina escalar (es decir, el número de fases en el pipeline), entonces, el tiempo necesario para ejecutar la misma operación secuencial sobre un vector de longitud n es:

$$Ts = knt$$

y el tiempo necesario para realizar la versión segmentada es:

$$Tp = kt + (n-1)t = (n + k - 1)t$$

De nuevo se obtiene que para $n > 1$, $Ts > Tp$

También se necesita un tiempo de arranque o startup. Éste es el tiempo necesario para conseguir que la operación se realice. Sobre una máquina secuencial, puede existir cierta penalización como consecuencia del tiempo necesario para preparar el lazo necesario para repetir la misma operación de punto flotante sobre un vector entero, ya que los elementos del vector también deben ser leídos desde la memoria. Si Ss es el número de unidades de tiempo t que componen el tiempo de arranque para la operación en un procesador secuencial, entonces Ts también debe incluir este tiempo. En ese caso:

$$Ts = (Ss + kn)t$$

En una máquina segmentada, el flujo desde los registros vectoriales o desde la memoria hacia el pipeline también necesita inicializarse. A este tiempo le denominamos Sp. En este caso existe también otra penalización de kt unidades de tiempo, necesarias para llenar inicialmente el pipeline. Por tanto, Tp debe incluir el tiempo de inicialización para la operación segmentada, es decir:

$$Tp = (Sp+k)t+(n-1)t$$

ó

$$Tp = (Sp+k+n-1)t$$

A medida que la longitud del vector aumenta (es decir, cuando n tiende a infinito), el tiempo de inicialización se vuelve despreciable en ambos casos. Esto significa que:

$$Ts - knt$$

mientras que

$$Tp - nt$$

Por tanto, para valores grandes de n, Ts es k veces mayor que Tp.
Existe un número de factores adicionales para describir el rendimiento de los ordenadores o procesadores vectoriales. A continuación se muestran algunas de estas medidas:

Rn : Para un procesador vectorial, el número de Mflops que se pueden obtener al operar sobre un vector de longitud n.

R¥: El número asintótico de Mflops que se pueden obtener sobre un ordenador vectorial a medida que la longitud del vector se va haciendo mayor. Esto significa que el tiempo de inicialización se haría completamente despreciable. Cuando los vectores son extremadamente largos, debería obtenerse un resultado del pipeline para cada t unidades de tiempo, ó para cada ciclo de reloj. De esta forma, el número de operaciones en punto flotante que se pueden completar en un segundo es de 1.0/t. Si se divide este resultado por un millón se obtiene el número de Mflops.

n1/2: La longitud, n, del vector tal que Rn es igual a R¥/2. De nuevo para vectores grandes, debería obtenerse un resultado a la salida del pipeline por cada t unidades de tiempo. De esta forma, n1/2 representa la longitud de vector necesaria para conseguir un resultado cada 2t unidades de tiempo ó cada dos ciclos de reloj.

nv: La longitud, n, de un vector tal que realizar una operación vectorial sobre los n elementos del vector resulta más eficiente que ejecutar las n operaciones escalares.

La figura 4 y la figura 5 muestran la relación existente entre estas cantidades para una máquina vectorial en la que t = 6 nanosegundos y Sp = 16t. Ésta es una representación teórica, ya que existen

ligeras variaciones en la curvatura de la gráfica en los casos en los que n es un múltiplo de la longitud del registro.

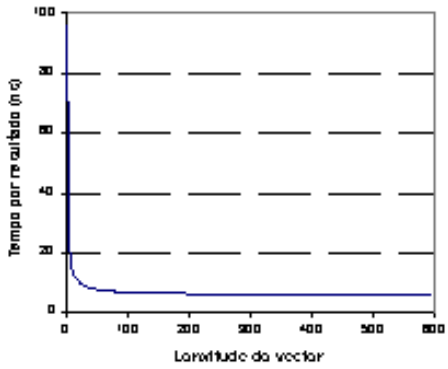

Figura 4. Rendemento vectorial representado polo tempo necesario para calcular un resultado en función da lonxitude do vector $t = 6$ ns, $S_\infty = 16t$.

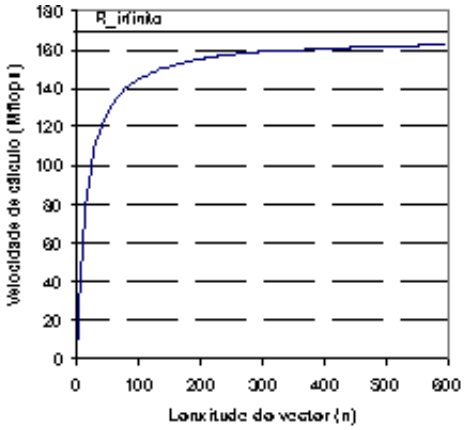

Figura 5. Rendemento vectorial R_n, expresado en Mflops por lonxitude do vector n. $t = 6$ns, $S_\infty = 16t$.

La tabla 4 proporciona algunas de las características de rendimiento de diversos ordenadores vectoriales. Los valores de R¥ y de n1/2 son los que se obtienen para la multiplicación de todos los elementos de dos vectores.

Programación de ordenadores vectoriales

Los ordenadores vectoriales pueden realizar operaciones en punto flotante sobre vectores de n elementos de forma más eficiente que sobre n variables escalares. Para poder crear programas que maximicen el rendimiento sobre estos ordenadores, es necesario organizar los datos en forma de vectores y utilizar las operaciones vectoriales tanto como sea posible.
Los diseñadores de ordenadores vectoriales a menudo proporcionan compiladores con construcciones adicionales al lenguaje diseñadas para ayudar al manejo de vectores o de matrices. Estas construcciones también ayudan al programador a pensar en un vector como en un entidad única, en lugar de pensar en él como una lista de elementos separados.

Los compiladores de los ordenadores vectoriales intentan convertir el código escalar en operaciones vectoriales (código vectorial) siempre que sea posible. A esta operación se le denomina vectorizar códigos, y en este caso, vectorización automática, cuando es el propio compilador el que determina las áreas del programa que se pueden vectorizar. Sin embargo, estos compiladores no conocen, ni el programa, ni los datos utilizados, tan bien como el propio programador, por lo que a veces son incapaces de vectorizar el código del mismo modo en que lo haría el propio programador. Por tanto, es recomendable introducir las directivas apropiadas para indicar cuáles son las operaciones vectoriales que se desean realizar, en lugar de dejar que sea el compilador el que las busque. A esta operación se la denomina vectorización manual, y es la base del porting de un programa a un ordenador vectorial.
¡Error!Marcador no definido.

Características de rendemento	Ano	Ciclo de reloxo (ns)	Rendemento pico (Mflops)	R_∞ ($x \times y$) (Mflops)	$n_{1/2}$ ($x \times y$)
Cray-1	1976	12.5	160	22	18
CDC Cyber 205	1980	20.0	100	50	86
Cray X-MP	1983	9.5	210	70	53
Cray-2	1985	4.1	488	56	83
IBM 3090	1985	18.5	108	54	70
ETA 10	1986	10.5	1750	-	-
Alliant FX/8	1986	1700	6	1	151
Cray C90	1990	4.2	952	-	650
Convex C3880	-	-	960	-	-
Cray 3-128	1993	2.1	948	-	-

Táboa 4. Características de rendemento dalgúns ordenadores vectoriais utilizando números en punto flotante de 64 bits. A expresión ($x \times y$) fai referencia á multiplicación de tódolos elementos de dous vectores x e y.