AT5118 – Arquitectura e Ingeniería de Computadores I

Juan Antonio Maestro

Universidad Antonio de Nebrija

Procesadores Vectoriales

Curso 2011-2012

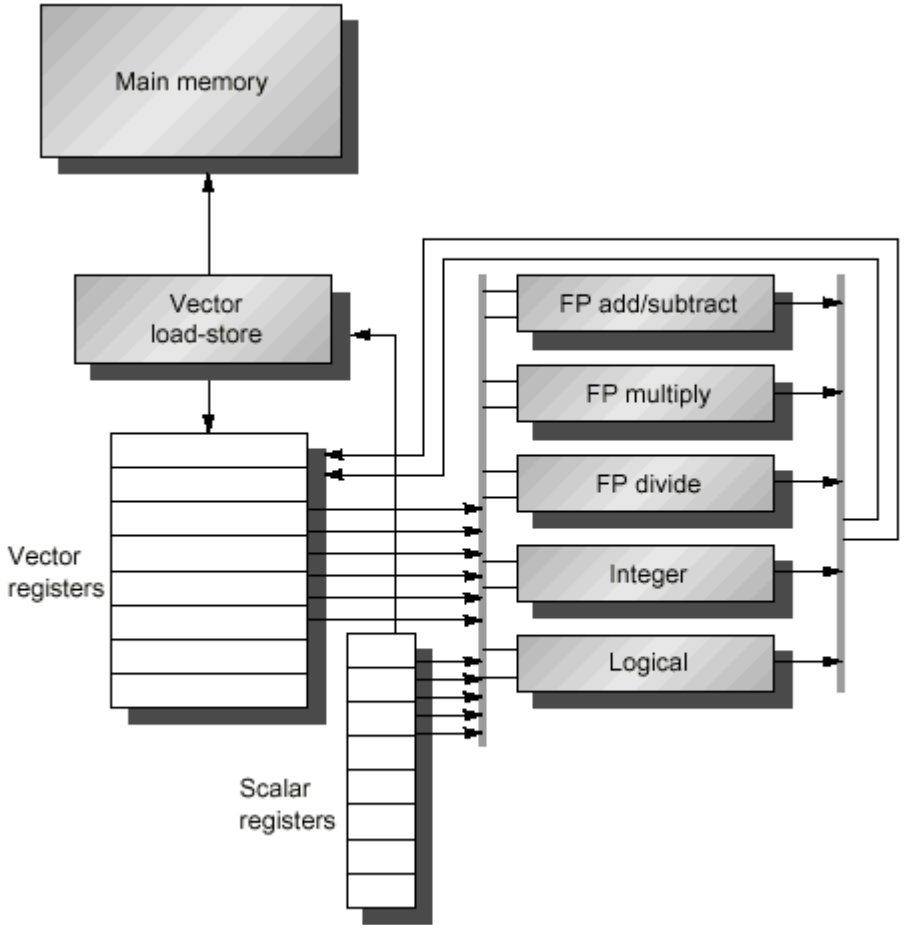
¿Por qué procesamiento vectorial?

- La segmentación tiene límites:
  - Si se aumenta mucho el número de etapas, baja el ciclo de reloj, pero aumentan las dependencias. Esto conlleva a un mayor CPI.
  - Velocidad de lectura de instrucciones: Es difícil traer instrucciones de memoria con una velocidad de lectura alta (cuello de botella de Flynn).

Ventajas de los procesadores vectoriales

- Proporcionan operaciones para trabajar con vectores. Una sola instrucción trabajo sobre todos los elementos de un vector:
  - El cálculo sobre los elementos de un vector suele ser independiente. Menos riesgos de datos.
  - El número de instrucciones es bajo. Se mitiga el cuello de botella de Flynn.
  - Los accesos a memoria siguen un patrón fijo. Los elementos de los vectores están ordenados.
  - Se eliminan bucles y dependencia de control.

DLX vectorial: DLXV



DLX vectorial: DLXV

- Registros vectoriales: Hay 8 registros para almacenar vectores de 64 elementos (cada elemento de 64 bits).
- Unidades Funcionales Vectoriales: Totalmente segmentadas. Aceptan una nueva operación cada ciclo. Hay 5 (3 en Punto Flotante y 2 enteras).
- Unidades Vectoriales de Carga y Almacenamiento: Totalmente segmentadas. También se ocupan de los accesos a memoria escalares.
- Registros escalares: Los del DLX. 32 GPR y 32 FPR de 32 bits.

Ejemplos de procesadores vectoriales

Processor	Year announced	Clock rate (MHz)	Registers	Elements per register (64-bit elements)	Functional units	Load-store units
CRAY-1	1976	80	8	64	6: add, multiply, reciprocal, integer add, logical, shift	1
CRAY X-MP CRAY Y-MP	1983 1988	120 166	8	64	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	2 loads 1 store
CRAY-2	1985	166	8	64	5: FP add, FP multiply, FP reciprocal/sqrt, integer (add shift, population count), logical	1
Fujitsu VP100/200	1982	133	8–256	32–1024	3: FP or integer add/logical, multiply, divide	2
Hitachi S810/820	1983	71	32	256	4: 2 integer add/logical, 1 multiply-add, and 1 multiply/divide–add unit	4
Convex C-1	1985	10	8	128	4: multiply, add, divide, integer/logical	1
NEC SX/2	1984	160	8 + 8192	256 variable	16: 4 integer add/logical, 4 FP multiply/divide, 4 FP add, 4 shift	8
DLXV	1990	200	8	64	5: multiply, divide, add, integer add, logical	1
Cray C-90	1991	240	8	128	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	4
Convex C-4	1994	135	16	128	3: each is full integer, logical, and FP (including multiply-add)	
NEC SX/4	1995	400	8 + 8192	256 variable	16: 4 integer add/logical, 4 FP multiply/divide, 4 FP add, 4 shift	8
Cray J-90	1995	100	8	64	4: FP add, FP multiply, FP reciprocal, integer/logical	
Cray T-90	1996	~500	8	128	8: FP add, FP multiply, FP reciprocal, integer add, 2 logical, shift, population count/parity	4

Operaciones típicas en el DLXV

Instruction	Operands	Function
ADDV ADDSV	V1,V2,V3 V1,F0,V2	Add elements of V2 and V3, then put each result in V1. Add F0 to each element of V2, then put each result in V1.
SUBV SUBVS SUBSV	V1,V2,V3 V1,V2,F0 V1,F0,V2	Subtract elements of V3 from V2, then put each result in V1. Subtract F0 from elements of V2, then put each result in V1. Subtract elements of V2 from F0, then put each result in V1.
MULTV MULTSV	V1,V2,V3 V1,F0,V2	Multiply elements of V2 and V3, then put each result in V1. Multiply F0 by each element of V2, then put each result in V1.
DIVV DIVVS DIVSV	V1,V2,V3 V1,V2,F0 V1,F0,V2	Divide elements of V2 by V3, then put each result in V1. Divide elements of V2 by F0, then put each result in V1. Divide F0 by elements of V2, then put each result in V1.
LV	V1,R1	Load vector register V1 from memory starting at address R1.
SV	R1,V1	Store vector register V1 into memory starting at address R1.
LVWS	V1,(R1,R2)	Load V1 from address at R1 with stride in R2, i.e., R1+i × R2.
SVWS	(R1,R2),V1	Store V1 from address at R1 with stride in R2, i.e., R1+i × R2.
LVI	V1,(R1+V2)	Load V1 with vector whose elements are at R1+V2(i), i.e., V2 is an index.
SVI	(R1+V2),V1	Store V1 to vector whose elements are at R1+V2(i), i.e., V2 is an index.
CVI	V1,R1	Create an index vector by storing the values 0, 1 × R1, 2 × R1, ..., 63 × R1 into V1.
S--V S--SV	V1,V2 F0,V1	Compare the elements (EQ, NE, GT, LT, GE, LE) in V1 and V2. If condition is true, put a 1 in the corresponding bit vector; otherwise put 0. Put resulting bit vector in vector-mask register (VM). The instruction S--SV performs the same compare but using a scalar value as one operand.
POP	R1,VM	Count the 1s in the vector-mask register and store count in R1.
CVM		Set the vector-mask register to all 1s.
MOVI2S MOVS2I	VLR,R1 R1,VLR	Move contents of R1 to the vector-length register. Move the contents of the vector-length register to R1.
MOVF2S MOVS2F	VM,F0 F0,VM	Move contents of F0 to the vector-mask register. Move contents of vector-mask register to F0.

Ejemplo: SAXPY / DAXPY

- Operación SAXPY (DAXPY en doble precisión):

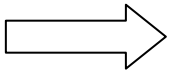
$$Y = a \cdot X + Y$$

Vectores de 64 elementos de 64 bits

DAXPY

```
        LD      F0,a
        ADDI    R4,Rx,#512
Loop:   LD      F2,0(Rx)
        MULTD   F2,F0,F2
        LD      F4,0(Ry)
        ADDD    F4,F2,F4
        SD      0(Ry),F4
        ADDI    Rx,Rx,#8
        ADDI    Ry,Ry,#8
        SUB     R20,R4,Rx
        BNEZ    R20,Loop
```

Vectorización



```
LD      F0,a
LV      V1,Rx
MULTSV  V2,F0,V1
LV      V3,Ry
ADDV    V4,V2,V3
SV      Ry,V4
```

Se reduce el número de instrucciones que se traen de memoria y los riesgos de datos

Operaciones vectoriales

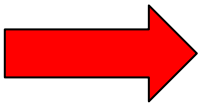
- Todas las operaciones vectoriales están segmentadas. En el DLXV, la suma tiene un tiempo de inicio de 6 ciclos, y la multiplicación de 7. Las operaciones aritméticas tienen latencia 1.
- Las cargas y almacenamientos en memoria también están segmentadas, con un tiempo de inicio de 12 ciclos. Pueden realizarse operaciones en paralelo, siempre que existan múltiples puertos de lectura y escritura en la memoria.

Operaciones vectoriales

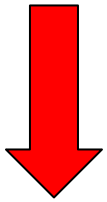
- Grupo: Conjunto de instrucciones vectoriales que pueden ejecutarse en paralelo. No pueden tener ningún tipo de riesgo entre ellas.
- Tiempo de grupo: Tiempo aproximado de ejecución de una instrucción vectorial. Es independiente del número de elementos del vector.
- m grupos con una longitud de vector de n, tardarían $m \cdot n$ ciclos. Esto no es exacto, ya que falta el tiempo de iniciación de las operaciones.

Ejemplo de operaciones vectoriales

DAXPY



```
LV      V1,Rx
MULTSV  V2,F0,V1
LV      V3,Ry
ADDV    V4,V2,V3
SV      Ry,V4
```



4 grupos:

1. **LV**
2. **MULTSV** **LV**
3. **ADDV**
4. **SV**

Tiempo de ejecución:

- 4 instantes + Iniciación de cada uno de los grupos.
- En este caso, no hay anticipación de datos

Strip-mining

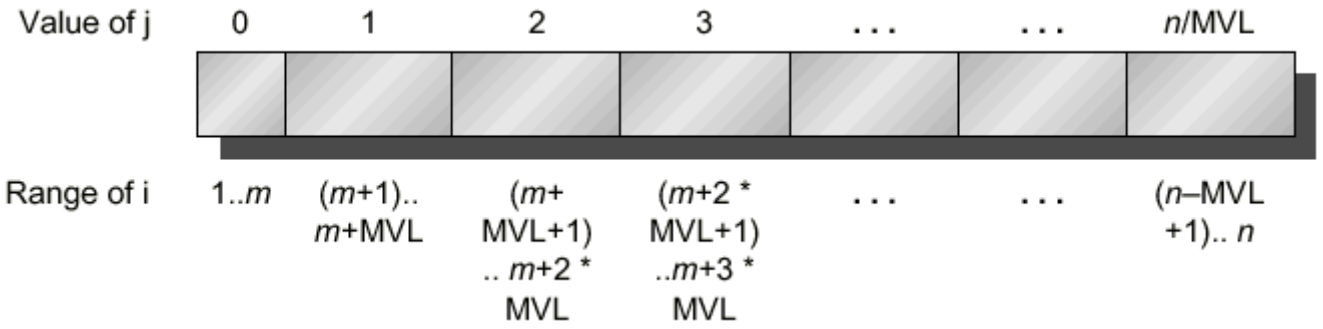
- Operación SAXPY (DAXPY en doble precisión):

$$Y = a \cdot X + Y$$

Vectores de 64 elementos de 64 bits

```
low = 0;
VL = n % MVL;
for (j=0;j<=(n/MVL);j++)
{
        for (i=low;i<=low+VL-1;i++)
                Y[i]= a*X[i] + Y[i];
        low = low + VL;
        VL = MVL;
}
```

⇒ Vectorizable



Strip-mining

- Tiempo de ejecución de un bucle con *strip-mining:*

$$T_n = \left\lceil \frac{n}{MVL} \right\rceil \cdot (T_{bucle} + T_{inicio}) + n \cdot T_{grupo}$$

  - n: número de elementos en cada vector
  - MVL: Longitud máxima del vector en la arquitectura
  - T_{bucle}: Tiempo de gestión del bucle
  - T_{inicio}: Tiempo de inicio de las operaciones vectoriales
  - T_{grupo}: número de grupos vectoriales

Tiempo de ejecución de DAXPY

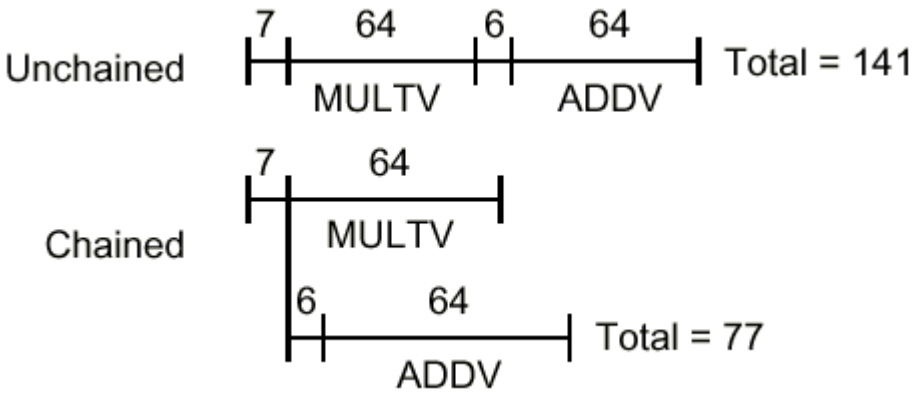
$$T_n = \left\lceil \frac{n}{MVL} \right\rceil \cdot (T_{bucle} + T_{inicio}) + n \cdot T_{grupo}$$

$$T_n = \left\lceil \frac{n}{64} \right\rceil \cdot (15 + 42) + 4 \cdot n$$

- MVL: 64
- T_{bucle}: 15 (estimado genérico)
- T_{inicio}: Suma de tiempos de inicio de operaciones
  (`LV+max(MULTSV,LV)+ADDV+SV` = 12+12+6+12).
- T_{grupo}: 4 grupos

Encadenamiento

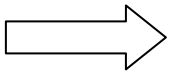
```
MULTV V1,V2,V3
ADDV  V4,V1,V5
```



Encadenamiento: La salida de una instrucción se anticipa a la entrada de la siguiente. El tiempo de ejecución viene dado por el número de elementos del vector (número de ciclos) más las iniciaciones de las instrucciones.

Operaciones vectoriales condicionales

```
for (i=0;i<64;i++)
{
        if (A[i] != 0)
                A[i] = A[i] - B[i];
}
```



```
LV      V1,Ra
LV      V2,Rb
LD      F0,#0
SNESV   F0,V1
SUBV    V1,V1,V2
CVM
SV      Ra,V1
```

Registro *Vector-Mask* (VM):

Vector de longitud MVL. Las operaciones vectoriales se aplicarán al elemento *i* del vector, si y sólo si el bit *i*-ésimo del VM está a 1

Nivel de vectorización

Nivel de vectorización sobre el benchmark *Perfect Club*:

Benchmark name	FP operations	FP operations executed in vector mode
ADM	23%	68%
DYFESM	26%	95%
FLO52	41%	100%
MDG	28%	27%
MG3D	31%	86%
OCEAN	28%	58%
QCD	14%	1%
SPICE	16%	7%
TRACK	9%	23%
TRFD	22%	10%

Medidas del rendimiento vectorial

- R_∞: Tasa de MFLOPS considerando vectores de longitud infinita.
- $N_{1/2}$: Longitud de vector necesaria para conseguir un rendimiento igual a la mitad de R_∞.
- N_v: Longitud de vector necesaria para que el modo de operación vectorial produzca un mejor rendimiento que el modo escalar.