Laboratorio de Arquitectura e Ingeniería de Computadores

PRÁCTICA III

COMPUTADORES VECTORIALES (DLXV)

Ingeniería de Computadores

Pedro Barquín Ayuso
Miguel Ballesteros García

1.-Realizar un programa que calcule la operación Y = a · X + Y donde X e Y son vectores de 100 elementos y a un escalar, primero con instrucciones escalares y después con instrucciones vectoriales. Comparar los resultados obtenidos. Hacer un estudio de las estadísticas obtenidas en el programa vectorial. (Todos los operandos deben ser de doble precisión).

Instrucciones escalares:

```
.data
vectorX:        .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

vectorY:        .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

a:              .double 1

.text
ini:

        addi r1,r0,#100
        addi r2,r0,#0

bucle:
        ld f2,vectorX(r2);
        ld f4,vectorY(r2);
        ld f0,a;

        multd f6,f0,f2;
        addd f4,f4,f6;

        sd vectorY(r2),f4;
        sub r1,r1,#1

        bnez r1,bucle;
        addi r2,r2,#8

trap #6
```

Hacemos la ecuación del enunciado utilizando una componente de cada vector en cada iteración y guardamos el resultado en el vector Y en su lugar correspondiente.

```
HARDWARE:
---------
Memoria: 65536 bytes, distribuida en 16 bancos.

Unidades escalares en Coma Flotante
 1 unidad(es) de suma/resta,            latencia =  2 ciclos
 1 unidad(es) de división,              latencia = 19 ciclos
 1 unidad(es) de multiplicación,        latencia =  5 ciclos

Unidades vectoriales
 1 unidad(es) de suma/resta,            latencia =  6 ciclos
 1 unidad(es) de división,              latencia = 20 ciclos
 1 unidad(es) de multiplicación,        latencia =  7 ciclos
 1 unidad(es) de carga/almac.,          latencia = 12 ciclos
 1 unidad(es) logica(s),                latencia =  1 ciclos

2/1 puertos de lectura/escritura en los regs. vectoriales

CONTADORES DE CICLOS:
---------------------
Ciclos de parada por dependencias:

  Entre instr. escalares         = 200
  Entre instr. de coma flotante  = 500
  Entre instr. vectoriales       = 0

Tiempo de ejecución del programa = 1603 ciclos

INSTRUCCIONES DE SALTO:
-----------------------

Total 100, tomados 99 (99.00%), no tomados 1 (1.00%)
```

Instrucciones vectoriales:

```
.data
vectorX:        .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1


vectorY:        .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1


a:              .double 1

.text
ini:




        ld f0,a
        addi r1,r0,#50
        addi r2,r0,#400
        movi2s vlr,r1


        lv v0, vectorX(r0)
        lv v1, vectorX(r2)
        lv v2, vectorY(r0)
        lv v3, vectorY(r2)
```

```
        multsv v4, f0, v0
        multsv v5, f0, v1

        addv v2,v2,v4
        addv v3,v3,v5

        sv vectorY(r0),v2
        sv vectorY(r2),v3

trap #6
```

Ahora en vez de ir componente a componente hacemos la ecuación de una vez utilizando el vector completo. De este modo no necesitamos utilizar un bucle

```
HARDWARE:
---------
Memoria: 65536 bytes, distribuida en 16 bancos.

Unidades escalares en Coma Flotante
 1 unidad(es) de suma/resta,            latencia =  2 ciclos
 1 unidad(es) de división,              latencia = 19 ciclos
 1 unidad(es) de multiplicación,        latencia =  5 ciclos

Unidades vectoriales
 1 unidad(es) de suma/resta,            latencia =  6 ciclos
 1 unidad(es) de división,              latencia = 20 ciclos
 1 unidad(es) de multiplicación,        latencia =  7 ciclos
 1 unidad(es) de carga/almac.,          latencia = 12 ciclos
 1 unidad(es) logica(s),                latencia =  1 ciclos

2/1 puertos de lectura/escritura en los regs. vectoriales

CONTADORES DE CICLOS:
---------------------
Ciclos de parada por dependencias:

  Entre instr. escalares         = 0
  Entre instr. de coma flotante  = 0
  Entre instr. vectoriales       = 355

Tiempo de ejecución del programa = 430 ciclos

INSTRUCCIONES DE SALTO:
-----------------------
No se han ejecutado instrucciones de salto.
```

2.-Realizar la parte vectorial del apartado 1 cambiando el número de bancos de memoria y, con vectores cuyas componentes no estén almacenadas en memora de forma consecutiva, analizar resultados. Realizar las pruebas y analizar los resultados para las siguientes situaciones.

Para este ejercicio hemos de plantear la ejecución del programa de forma que cumplamos las restricciones de la tabla por lo que contaremos con 3 códigos con los cuales hemos de realizar 3 simulaciones para cada una de las cantidades de bancos de memoria disponibles.

Bancos de memoria	Distancia en memoria entre componentes
64 bancos	Posiciones consecutivas
	.space 8
	.space 24
32 bancos	Posiciones consecutivas
	.space 8
	.space 24
16 bancos	Posiciones consecutivas
	.space 8
	.space 24

Posiciones consecutivas:

```
.data
        x:      .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

        y:      .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

        a:      .double 5

.text

ini:    ld f0,a
        addi r1,r0,#50
        addi r2,r0,#400         ;400 ya que usar la mitad de ellos ocupando 8 nos lleva a 8*50=400 que es donde
empieza r2
        movi2s vlr,r1           ;Ponemos a 50 el número máximo de datos en el registro vectorial la mitad

        lv v0, x(r0)            ;empieza desde el 0
        lv v1, x(r2)            ;empieza desde la mitad que es 400
        lv v2, y(r0)
        lv v3, y(r2)

        multsv v4, f0, v0
        multsv v5, f0, v1

        addv v2,v2,v4
        addv v3,v3,v5

        sv y(r0),v2
        sv y(r2),v3

trap #6
```

Posiciones separadas 8:

```
.data
        x:      .double 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
                .double 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
                .double 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
                .double 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
                .double 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
        y:      .double 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
                .double 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
                .double 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
                .double 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
                .double 1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0,1,0
        a:      .double 5

.text
ini:    ld f0,a
        addi r1,r0,#50
        addi r5,r0,#16          ;separación entre datos
        movi2s vlr,r1           ;Ponemos a 50 el número máximo de datos por operación

        addi r6,r0,x            ;sacamos la dirección de empiece del vector x
        lvws v0,r6,r5;          ;sacamos 50 elementos
        addi r6,r6,#800         ;sacamos la dirección de empiece del vector x a la mitad
        lvws v1,r6,r5;          ;sacamos otros 50 elementos

        addi r6,r0,y            ;sacamos la dirección de empiece del vector x
        lvws v2,r6,r5;          ;sacamos 50 elementos
        addi r6,r6,#800         ;sacamos la dirección de empiece del vector x a la mitad
        lvws v3,r6,r5;          ;sacamos otros 50 elementos

        multsv v4, f0, v0       ;multiplicamos por el escalar
        multsv v5, f0, v1
```

```
	addv v2,v2,v4			;sumamos vector a vector
	addv v3,v3,v5

	addi r6,r0,y	;almacenamos en las posiciones en las que hemos extraído los datos iniciales para guardar la referencia
	svws r6,r5,v2
	addi r6,r6,#800
	svws r6,r5,v3
trap #6
```

Posiciones separadas 24:

```
.data
	x:	.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
	y:	.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
		.double 1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0,1,0,0,0
	a:	.double 5

.text
```

```
ini:    ld f0,a
        addi r1,r0,#50
        addi r5,r0,#32          ;separación entre datos
        movi2s vlr,r1           ;Ponemos a 50 el número máximo de datos por operación

        addi r6,r0,x            ;sacamos la dirección de empiece del vector x
        lvws v0,r6,r5;          ;sacamos 50 elementos
        addi r6,r6,#1600        ;sacamos la dirección de empiece del vector x a la mitad
        lvws v1,r6,r5;          ;sacamos otros 50 elementos

        addi r6,r0,y            ;sacamos la dirección de empiece del vector x
        lvws v2,r6,r5;          ;sacamos 50 elementos
        addi r6,r6,#1600        ;sacamos la dirección de empiece del vector x a la mitad
        lvws v3,r6,r5;          ;sacamos otros 50 elementos

        multsv v4, f0, v0       ;multiplicamos por el escalar
        multsv v5, f0, v1

        addv v2,v2,v4           ;sumamos vector a vector
        addv v3,v3,v5

        addi r6,r0,y    ;almacenamos en las posiciones en las que hemos extraído los datos iniciales para guardar la referencia
        svws r6,r5,v2
        addi r6,r6,#1600
        svws r6,r5,v3

trap #6
```

Tabla con los resultados en función del banco de registros y espaciado entre datos:

Banco/espacio	**Nada**	**8**	**24**
16	430	553	955
32	430	550	953
64	430	550	953

Conclusión En una máquina vectorial, los accesos no son a datos individuales, sino a colecciones de ellos (vectores), Si el espaciado es 1, proporcionan el mejor rendimiento. Sin embargo, con espaciado no unidad, puede no alcanzarse el rendimiento óptimo. Ejemplo: si el espaciado es 8, y hay 8 bancos, todos los accesos van al mismo banco (=>secuencialización). Por ello en nuestro ejercicio observamos que sin espaciado el rendimiento es el mismo en los tres casos pero al aumentar el espaciado podemos apreciar que hay un decremento entre el uso de 16 a (32/64) y esto es porque la forma de aprovechar los bancos con una separación entre datos aumenta (reduce el número de ciclos) al aumentar el número de bancos de memoria

3.-Realizar un programa que multiplique dos matrices de 10 x 10 usando instrucciones vectoriales. Donde A, B son las matrices a multiplicar, C la matriz producto y n el orden de las matrices. Por lo tanto, se trata de reiterar una operación del tipo DAXPY como la comentada en el principio de la práctica.

```
.data
        x:      .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

        y:      .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
                .double 1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1

        z:      .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
                .double 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

        cero:   .double 0,0,0,0,0,0,0,0,0,0
.text

ini:    addi r31,r0,#10
        movi2s vlr,r31
        addi r1,r0,#0
        addi r3,r0,#0
        addi r30,r30,#10

bucle1: addi r29,r29,#10
        addi r2,r0,#0
        lv v1,cero(r0)

bucle2: ld f0,x(r1)
        lv v0,y(r2)

        multsv v0,f0,v0
        addv v1,v0,v1
```

```
        addi r1,r1,#8
        subi r29,r29,#1
        bnez r29,bucle2
        addi r2,r2,#80

        sv z(r3),v1
        subi r30,r30,#1
        bnez r30,bucle1
        addi r3,r3,#80
trap#6
```

Ponemos el tamaño del vector, los contadores de los bucles e inicializamos los registros para el desplazamiento de los vectores. Creamos un vector “cero” para poner a 0 el vector v1 en cada iteración.

Siguiendo la sugerencia del enunciado de la práctica creamos dos bucles. El bucle2 carga el elemento X(ij) y el vector Y(j), los multiplica y lo va sumando con el vector que ha quedado de la suma de iteración anterior desplázanos hacia la derecha en el primer vector de la matriz X y va bajando a cada fila de la matriz Y. Una vez que salimos de ese bucle seguimos en el bucle1 que va bajando de fila en la matriz X y guarda el vector resultante en la matriz Z.

```
HARDWARE:
---------
Memoria: 65536 bytes, distribuida en 16 bancos.

Unidades escalares en Coma Flotante
 1 unidad(es) de suma/resta,            latencia =  2 ciclos
 1 unidad(es) de división,              latencia = 19 ciclos
 1 unidad(es) de multiplicación,        latencia =  5 ciclos

Unidades vectoriales
 1 unidad(es) de suma/resta,            latencia =  6 ciclos
 1 unidad(es) de división,              latencia = 20 ciclos
 1 unidad(es) de multiplicación,        latencia =  7 ciclos
 1 unidad(es) de carga/almac.,          latencia = 12 ciclos
 1 unidad(es) logica(s),                latencia =  1 ciclos

2/1 puertos de lectura/escritura en los regs. vectoriales

CONTADORES DE CICLOS:
---------------------
Ciclos de parada por dependencias:

  Entre instr. escalares         = 110
  Entre instr. de coma flotante  = 0
  Entre instr. vectoriales       = 4135

Tiempo de ejecución del programa = 5137 ciclos

INSTRUCCIONES DE SALTO:
-----------------------

Total 110, tomados 99 (90.00%), no tomados 11 (10.00%)
```