Guía de Trabajos Prácticos: Paralelismo con F95/HPF/OMP

En los siguientes ejercicios se consideran:

i) códigos seriales en *Fortran* 95 (F95), con diversas operaciones matriciales;

ii) paralelismo con *High Perfomance Fortran* (HPF) en equipos con memoria distribuida tal como lo es un *cluster Beowulf*;

iii) paralelismo con F95/C++ y OMP en equipos con memoria distribuida, por ejemplo, sobre una Pentuam dual.

Para resolverlos se sugiere escribir/correr programas de prueba. En el caso del F95 los programas fuente pueden ser o bien formateados (con extensión *.f) o bien con formato libre (extensión *.f90).

- En el caso de contar con una PC con un único procesador (es decir que no disponemos, por ejemplo, de una Pentium dual) claro está que sólo podrá compilarse en forma serial. Entre los compiladores de libre distribución bajo Linux que soportan OMP destacamos: `KAI-INTEL`[3] tanto para F95 como para C++. Por otra parte, `g++` suele venir pre-instalado en los paquetes de distribución, mientras que `g95`[2] es un desarrollo muy reciente y hay que bajarlo de la red. Empero, ninguno de los dos soporta ninguna clase de paralelismo. Para usarlos puede emplearse los comandos

```
g++ -o test.exe test.cpp
/opt/intel_cc_80/bin/icc   -o test.exe test.cpp
g95 -o test.exe test.f90
/opt/intel_fc_80/bin/ifort -o test.exe test.f90
```

- En el caso del `cluster Beowulf` "Gerónimo" del CIMEC, se disponen de los compiladores PGI[5] y ADAPTOR[1] para los programas en HPF (en caso de PGI puede convenir *linkear* con MPI):

```
pghpf -V -Minfo -Mautopar -Mmpi -fast -o test.exe test.hpf
adaptor -free -o test.exe test.f90
```

Ejercicio 1.1 Usando las intrínsecas matriciales `LBOUND`, `UBOUND`, `SIZE` y `SHAPE`, determinar los extremos inferior y superior, el tamaño y la forma de cada uno de los siguientes arreglos:

```
REAL   , DIMENSION (1:10)     :: a
REAL   , DIMENSION (2,0:2)    :: b
INTEGER, DIMENSION (-1:1,3,2) :: c
REAL   , DIMENSION (0:1,3)    :: d
```

Ejercicio 1.2 . Para un arreglo `a` de enteros en 2D, escribir una instrucción matricial `WHERE` en donde se anulan todas sus entradas `a (i,j)` impares.

Ejercicio 1.3 . Para un arreglo `a` de reales en 2D, escribir una instrucción matricial `WHERE ELSE WHERE` en donde, si la entrada `a (i,j)` es positiva, entonces calcular su raíz cuadrada, sino anularla.

Ejercicio 1.4 . Para los arreglos

$$\texttt{a} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} \quad ; \quad \texttt{b} = \begin{bmatrix} 9 & 8 & 2 \\ 6 & 5 & 4 \\ 3 & 2 & 1 \end{bmatrix} ; \tag{1}$$

determinar el resultado de las siguientes instrucciones de desplazamiento circular y lineal: (i) `CSHIFT(a,SHIFT=-1)`; (ii) `CSHIFT (a,SHIFT=+3)`; (iii) `EOSHIFT (b,SHIFT=+1,DIM=1)`; (iv) `EOSHIFT (b,SHIFT=-1,DIM=2)`.

Ejercicio 1.5 . Para el arreglo

$$\texttt{a} = \begin{bmatrix} 3 & -4 & 8 & 6 \\ 1 & 3 & 5 & 4 \\ 0 & 2 & -5 & -1 \end{bmatrix} ; \tag{2}$$

determinar el resultado de las siguientes instrucciones de reducción: (i) `MAXLOC(a)`; (ii) `MAXVAL(a)`; (iii) `MINLOC(a,a<5)`; (iv) `MINVAL(a,a<5)`; (v) `COUNT(a,a>0)`; (vi) `COUNT(a,a<0)`.

Ejercicio 1.6 . Reacomodar el vector

$$\texttt{v} = \begin{bmatrix} 3 & 1 & 0 & -4 & 3 & 2 & 8 & 5 & -5 & 6 & 4 & -1 \end{bmatrix} ; \tag{3}$$

como el arreglo

$$\texttt{a} = \begin{bmatrix} 3 & -4 & 8 & 6 \\ 1 & 3 & 5 & 4 \\ 0 & 2 & -5 & -1 \end{bmatrix} ; \tag{4}$$

mediante la instrucción `RESHAPE (v, forma)`, donde `forma` es un vector con el número de filas y columnas del arreglo `a`.

Ejercicio 1.7 . Determinar el efecto de la instrucción `MERGE (a, b, c)`, cuando

$$\texttt{a} = \begin{bmatrix} 2 & 4 & 6 \\ 8 & 6 & 4 \end{bmatrix} \quad ; \quad \texttt{b} = \begin{bmatrix} 1 & 3 & 5 \\ 7 & 9 & 3 \end{bmatrix} \quad ; \quad \texttt{c} = \begin{bmatrix} T & T & F \\ F & T & F \end{bmatrix} . \tag{5}$$

Ejercicio 1.8 . Considere las matrices

$$\texttt{a} = \begin{bmatrix} 0 & 5 & 0 \\ 1 & 0 & 9 \end{bmatrix} \quad ; \quad \texttt{b} = \begin{bmatrix} 7 & -2 & 0 \\ -3 & -4 & 5 \end{bmatrix} ; \tag{6}$$

y los vectores

$$\texttt{v} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 \end{bmatrix} \quad ; \quad \texttt{z} = \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} ; \tag{7}$$

determinar el resultado de las siguientes instrucciones: (i) `PACK (a, a.NE.0)`; (ii) `PACK (b, b>0)`; (iii) `PACK (a, a.NE.0, VECTOR=v)`; (iv) `PACK (b, a>0, VECTOR=z)`.

Ejercicio 1.9 . Considere los arreglos

$$\texttt{a} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad ; \quad \texttt{b} = \begin{bmatrix} F & T & F \\ T & F & F \\ F & F & T \end{bmatrix} \quad ; \quad \texttt{c} = \begin{bmatrix} 4 & 5 & 6 & 7 & 8 \end{bmatrix} ; \tag{8}$$

determinar el resultado de las siguientes instrucciones: (i) `UNPACK (c, MASK = b, FIELD = -3)`; (ii) `UNPACK (c, MASK = b, FIELD = a)`.

Ejercicio 1.10 . Considere el arreglo `a` = [11 22 33 44 55], de tamaño `n = SIZE (a)`. Determine el resultado de las siguientes instrucciones:

```
DO      i = 2 , n  ; a (i) = a (i-1) ; END DO
FORALL (i = 2 : n)   a (i) = a (i-1)
```

Ejercicio 1.11 . Escriba una función matricial F90 que devuelva el producto exterior de los vectores `a` y `b`, de tamaño `n = SIZE (a) = SIZE (b)` [Indicación: use la intrínseca `SPREAD`. El resultado es la matriz `c` tal que `c (i,j) = a (i) * b (j)`, para todo `i,j`].

Ejercicio 1.12 . Determinar el número de comunicaciones entre procesadores para las siguientes instrucciones `FORALL`:

```
       INTEGER, DIMENSION (16)       :: a, b, c
       INTEGER, DIMENSION (32)       :: d
!HPF$  PROCESSORS, DIMENSION (4)     :: p
!HPF$  DISTRIBUTE (BLOCK)  ONTO  p   :: a, b, d
!HPF$  DISTRIBUTE (CYCLIC) ONTO  p   :: c
       INTEGER                       :: i
       FORALL (i=1:16)  a (i) = b (i)
       FORALL (i=1:16)  c (i) = a (i)
       FORALL (i=1:15)  a (i) = b (i+1)
       FORALL (i=1:16)  a (i) = d (2*i-1)
```

Ejercicio 1.13 . Para la matriz `a` = diag (1 4 9 16 25 36 49 64), determinar la salida del siguiente fragmento de código

```
FORALL (i=1:8)
  a (i,i) = SQRT ( a (i,i) )
  FORALL (j=i-3:i+3, j .NE. i .AND. j > 0 .AND. j < 9)
    a (i,j) = a (i,i) + a (j,j)
  ENDFORALL
ENDFORALL
```

Ejercicio 1.14 . Escriba un procedimiento HPF para trasponer una matriz `a` de tamaño `n` $\times$ `n` usando la instrucción FORALL.

Ejercicio 1.15 . Dado el arreglo $\boldsymbol{a} = (a_1, a_2, ..., a_n)$ escriba un procedimiento HPF que devuelva otro arreglo $\boldsymbol{b} = (b_1, b_2, ..., b_n)$ con el promedio de los elementos adyacentes de $\mathbf{a}$, i.e. $b_i = (a_{i-1} + a_{i+1})/2$, para todo i. Para b_1 usar el segundo y el último elemento de $\boldsymbol{a}$, mientras que para b_n usar el primer y penúltimo elemento de $\boldsymbol{a}$.

Ejercicio 1.16 . Escriba un procedimiento HPF para calcular el producto matriz vector $\boldsymbol{y} = \boldsymbol{A}\boldsymbol{x}$. Use intrínsecas donde sea posible. Hágalo sobre un arreglo de procesadores abstracto p en 1D y en 2D. Cúal es mejor y argumente por qué.

Ejercicio 1.17 . [Uso práctico de punteros]. Escriba un programa F95 que para hallar la raíz cuadrada aproximada $\boldsymbol{x}$ de un arreglo de elementos reales positivos $\boldsymbol{c}$ mediante la fórmula iterativa $\boldsymbol{x}_k = (\boldsymbol{x}_{k-1} + \boldsymbol{c}./\boldsymbol{x}_{k-1})/2$, para $k = 1, 2, ...$, siendo $\boldsymbol{x}_0$ una cierta aproximación inicial. El programa debe incluir el uso de punteros, arreglos dinámicos y funciones matriciales.

Ejercicio 1.18 . Escribir un procedimiento HPF que, dada una matriz banda en formato comprimido A_c de $n \times m$, y el vector x de $n \times 1$, devuelva otro vector b con el producto matriz-vector, usando intrínsecas tales como `EOSHIFT`, `SPREAD`, `SUM` o `SIZE`, donde $m = m_1 + 1 + m_2$ es el ancho de la banda, primero con las m_1 sub-diagonales, luego la diagonal principal y, finalmente, las m_2 supra-diagonales, como columnas consecutivas en la matriz en formato comprimido A_c.

Ejercicio 1.19 . Escribir un procedimiento HPF para multiplicar dos matrices A y B de tamaño $n \times n$ (i) sin usar la función matricial intrínseca `MATMUL`, y (ii) usándola. Compare el rendimiento de su programas de multiplicación de matrices. Explique las posibles diferencias.

Ejercicio 1.20 . Escribir un programa HPF para resolver un Sistema de Ecuaciones Algebraicas Lineales (SEAL) mediante un método directo en paralelo, en el cual se llama a un procedimiento de eliminación de Gauss encapsulado en un módulo. Experimentar con las distribuciones `(block,*)`, `(*,block)`, `(block,block)`, `(cyclic,*)`, `(cyclic,cyclic)`, `(cyclic,block)`, comparando el rendimiento de cada una de ellas. Explique las posibles diferencias.

Ejercicio 1.21 . Escribir un programa HPF para resolver un Sistema de Ecuaciones Algebraicas Lineales (SEAL) mediante el método iterativo de Gauss-Seidel. Experimentar con las diferentes distribuciones posibles. El programa debe incluir el uso de punteros, arreglos dinámicos y funciones matriciales.

Ejercicio 1.22 . Escriba un programa HPF para calcular el conjunto de Mandelbrot usando codificación matricial donde fuera posible.

Ejercicio 1.23 . Escriba un programa HPF para el "juego de la vida". El mismo consiste en una malla 2D de células, las cuales tienen dos estados posibles: vivas o muertas. En cada iteración el nuevo estado de cada célula se determina por el de su primera capa de células vecinas de la iteración previa. El código deberá inicializar el tablero y luego, iterando, (i) calcula el número de células vivas en la primera capa de vecinas (`CSHIFT`): si es 3 entonces nace, si fuera menor a 2 o mayor a 3 entonces muere, sino sin cambios.

Ejercicio 1.24 (Opcional OMP con C++/F95). Transcriba y experimente los ejemplos de OMP con C++ dados en la Norma OMP/C++[4] y a continuación transcríbalos en OMP/F95.

Ejercicio 1.25 (Opcional OMP con C++/F95). Transcriba y experimente los ejemplos incluidos en la página del curso, referidos a una misma tarea efectuada tanto con C++ como con F95.

Ejercicio 1.26 [Opcional para Ingeniería Informática (II)]. Bajar y configurar en un *cluster Beowulf* el sistema de colas `TORQUE`[6] (*Tera-scale Open-source Resource and QUEue manager*), el cual es un administrador de recursos en colas *batch* para *cluster Beowulf*. Este paquete se apoya e incluye al sistema de colas `OpenPBS` 2.3.12. La configuración debe ser tal que los nodos deben ser de tipo *time-sharing* (de recurso compartido entre las diferentes colas).

Referencias

[1] T. Brandes. ADAPTOR: Parallel fortran compilation system. http://www.scai.fraunhofer.de/EP-CACHE/adaptor/www/adaptor_home.html.

[2] G95: the GNU Fortran 95 compiler. http://g95.sourceforge.net/.

[3] KAI-Intel: C++ and Fortran 95 compilers. http://www.kai.com/.

[4] Open Machine Parallel: Fortran (v. 2.0, november 2000) and c++ (v. 2.0, march 2002). http://www.openmp.org/.

[5] Portland Cluster Kit: C++/F95/HPF compilers. http://www.pgroup.com/.

[6] Tera-scale Open-source Resource and QUEue manager. http://www.supercluster.org/projects/torque.