Computación de Altas Prestaciones

Seminario 5

1

Contenido

1. Introducción a los archivos MEX.
2. Configuración de los archivos MEX.
3. Invocación a funciones de Matlab (recordatorio).
4. Creación de archivos MEX en C.
5. Ficheros MEX y OpenMP.
6. Depuración de archivos MEX.

2

1. Introducción a los archivos MEX

- Matlab permite llamar a funciones escritas en C, C++ o Fortran.
- Se pretende obtener mayor eficiencia o reutilizar código ya existente.
- Matlab ofrece una herramienta para generar un ejecutable con las funciones a ser usadas desde el propio Matlab.
- El archivo generado tiene extensión .mex (ejecutable de Matlab).
- La extensión depende del S.O.: Windows (*.mexw32, *.mexw64), Linux (*.mexglx, *.mexa64) o Mac OS X (*.mexmaci).

3

2. Configuración de los archivos MEX

- Para poder crear archivos MEX necesitamos instalar compiladores compatibles con la versión de Matlab que estemos usando.
- Compiladores soportados: https://es.mathworks.com/support/requirements/previous-releases.html
- Para Matlab R2021a, en Windows: MinGW, Microsoft Visual C++ o Intel Parallel Studio XE.

MATLAB Product Family – Release 2021a

Compiler	MATLAB For MEX-file compilation, loadlibrary, C++ interface, and external usage of MATLAB Engine and MAT-file APIs	MATLAB Coder For all features	GPU Coder For all features	SimBiology For accelerated computation	Fixed-Point Designer For accelerated computation	HDL Coder For accelerated testbench simulation	HDL Verifier For DPI and TLM component generation	Audio Toolbox For validating and generating audio plugins	ROS Toolbox For custom messages and code generation
MinGW 6.3 C/C++ (Distribution: mingw-w64) *Download Now* Available at no charge	✓	✓		✓	✓	✓	✓		
Microsoft Visual C++ 2019 product family	✓	✓	✓	✓	✓			✓	

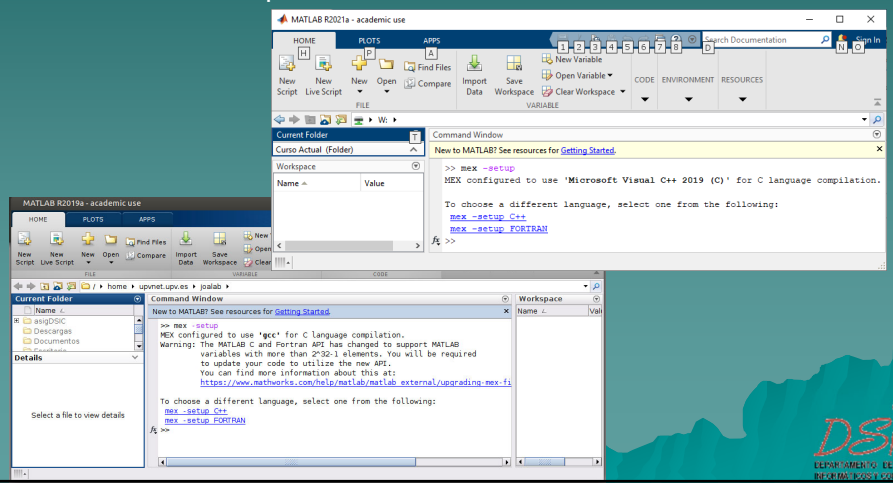
- Adicionalmente:
  - Linux 64 bits: compiladores de GNU (gcc, etc.) a partir de 7.x.
  - Mac OS X: compiladores Xcode 11.x o 12.x, Intel Parallel Studio XE.

4

2. Configuración de los archivos MEX

- Para averiguar qué compiladores tenemos disponibles, escribiremos:

  >> mex -setup



5

3. Invocación a funciones en Matlab

- A una misma función de Matlab se le puede llamar con un número variable de argumentos de entrada y/o de salida.
- Ejemplos:

  % Devuelve el máximo del vector x:

  >> maximo=max(x)

  % Devuelve el máximo del vector *x* y la posición en la que se encuentra:

  >> [maximo, posicion]=max(x)

  % Devuelve un vector en el que se almacena el máximo de los elementos de los vectores *x* e *y*, comparando los elementos que ocupan idénticas posiciones:

  >> maximo=max(x,y)

6

4. Creación de archivos MEX en C

- El objetivo es invocar a una función escrita en C como si se tratara de una función de Matlab.
- Componentes de un archivo MEX programado en C:
  - Una o varias funciones en C, a las que se quiere invocar.
  - La función *mexFunction*, la cual actúa como interfaz entre las funciones en C y Matlab. Fundamentalmente, controla el paso de parámetros de entrada y salida.
- Para llamar desde Matlab a la función, utilizaremos el nombre del archivo MEX sin la extensión.

4. Creación de archivos MEX en C

- Ejemplo 1. Queremos invocar desde Matlab a la siguiente función escrita en C:

```
void pordos (double y[], double x[]) {
    y[0] = 2.0*x[0];
}
```

- La llamada desde Matlab debería ser de alguno de estos dos modos:

```
>> pordos(4)
ans=8
>> d=pordos(4)
d=8
```

4. Creación de archivos MEX en C

- Creación del archivo MEX:
  - En primer lugar, incorporamos el archivo *mex.h* y escribimos la función *pordos* en el archivo *pordos.c*.

```
#include "mex.h"

void pordos(double y[], double x[]) {
      y[0] = 2.0*x[0];
}
...
```

9

4. Creación de archivos MEX en C

- A continuación escribimos la función *mexFunction*, cuya cabecera es siempre idéntica:

```
void mexFunction (int nlhs, mxArray *plhs[], int nrhs,
                  const mxArray *prhs[])
```

- Argumentos:
  - Dos enteros indicando el número de argumentos de entrada (*nrhs*) y de salida (*nlhs*) con los que se llamará a la función desde Matlab.
  - Dos vectores de punteros, de longitud nrhs y nlhs, a los argumentos de entrada (*prhs*) y de salida (*plhs*).

    El tipo *mxArray* es un tipo de datos propio que engloba a cualquier vector de Matlab.

10

4. Creación de archivos MEX en C

- Declaración de variables:
  - En primer lugar, declaramos las variables que vamos a necesitar: dos punteros *x* e *y* a reales de doble precisión para almacenar los parámetros de entrada y salida.
  - Dos variables *mrows* y *ncols* de tipo *size_t* (enteros) para comprobar que la dimensión del dato de entrada es correcta.

```
...
double *x,*y;
size_t mrows, ncols;
...
```


4. Creación de archivos MEX en C

- Comprobación del número de argumentos:
  - Debemos comprobar que, en este caso, *nrhs* y *nlhs* deben valer 1. En caso contrario, mostraremos un mensaje de error con la función *mexErrMsgIdAndTxt*.

```
...
/* Check for proper number of arguments. */
if (nrhs!=1) {
    mexErrMsgIdAndTxt ("MATLAB:pordos:invalidNumInputs",
                       "One input required.");
}
else if (nlhs>1) {
    mexErrMsgIdAndTxt ("MATLAB:pordos:maxlhs",
                       "Too many output arguments.");
}
...
```

4. Creación de archivos MEX en C

- Comprobamos las dimensiones y el tipo del argumento de entrada prhs[0]:
  - Obtenemos sus dimensiones mediante las funciones *mxGetM* y *mxGetN*.
  - Comprobamos el tipo de datos de prhs[0], mediante las funciones *mxIsDouble* y *mxIsComplex*, y que se trata de un número real de dimensiones 1 por 1.

```
...
 /* The input must be a noncomplex scalar double.*/
 mrows = mxGetM (prhs[0]);
 ncols = mxGetN (prhs[0]);

 if (!mxIsDouble (prhs[0]) || mxIsComplex (prhs[0]) ||
    !(mrows==1 && ncols==1) ) {
        mexErrMsgIdAndTxt ("MATLAB:pordos:inputNotRealScalarDouble",
                           "Input must be a noncomplex scalar double.");
 }
...
```


4. Creación de archivos MEX en C

- Dimensionamos las variables de salida:
  - Habitualmente usaremos la función *mxCreateDoubleMatrix*, a la que le pasaremos el número de filas y de columnas y el tipo de dato del que consta.
  - Creamos el "enlace", mediante punteros, de los argumentos prhs[0] y plhs[0] de la función *MexFunction* con los argumentos *x* e *y* de la función *pordos*.

```
...
 /* Create matrix for the return argument. */
 plhs[0] = mxCreateDoubleMatrix((mwSize)mrows,(mwSize)ncols, mxREAL);

/* Assign pointers to each input and output. */
x = mxGetPr(prhs[0]);
y = mxGetPr(plhs[0]);
...
```

4. Creación de archivos MEX en C

- Invocamos a la función pordos:

```
...

    /* Call the pordos subroutine */
    pordos(y,x);
}
```

- Una vez guardado el archivo *pordos.c* en el directorio de trabajo de Matlab, lo compilamos:

  >> mex pordos.c

- Al compilarlo se genera un archivo con el mismo nombre y con extensión *.mexw64 (en Windows, con 64 bits) o *.mexa64 (en Linux, con 64 bits).

15

4. Creación de archivos MEX en C

- Ejemplo 2. Permite trabajar con vectores y escalares en archivos MEX:

```
void arrayProduct(double x,double *y,double *z,mwSize n)
{
    mwSize i;
    /* multiply each element y by x */
    for (i=0; i<n; i++)
        z[i] = x * y[i];
  }
```

- La llamada desde Matlab debería ser así:

```
>> z = arrayProduct(2, [3 4 8])

z = 6 8 16
```

16

4. Creación de archivos MEX en C

```
ncols = mxGetN(prhs[1]);
/* Create the output matrix */
plhs[0]=mxCreateDoubleMatrix(1,(mwSize)ncols,mxREAL);

/* Get a pointer to the real data in the output matrix */
outMatrix = mxGetPr(plhs[0]);

/* Call the computational routine */
arrayProduct(multiplier,inMatrix,outMatrix,(mwSize)ncols);
```

17

4. Creación de archivos MEX en C

- Ejercicio 1. Producto de matrices:
  - Crea la función *mexFunction* para la siguiente función *prodmat*, encargada de calcular el producto de dos matrices cuadradas.
  - Toma tiempos, comparándolos con la versión *jki* que escribiste en Matlab en el primer seminario. Trabaja con matrices de tamaño 2000 x 2000.

```
void prodmat (double *A, double *B, double *C, mwSize n) {
   mwSize i, j, k;
   for (j=0; j<n; j++)
       for (k=0; k<n; k++)
          for (i=0; i<n; i++)
             C[i+j*n]= C[i+j*n]+ A[i+k*n]* B[k+j*n];
}
```

18

5. Ficheros MEX y OpenMP

- Aunque no está soportado oficialmente, OpenMP funciona con archivos MEX.
- Para poder compilar ficheros MEX con funciones o directivas de OpenMP debemos usar la siguiente línea de compilación:
  - Windows:
    >> mex 'OPTIMFLAGS= /openmp /O2 /Oy-' mi_programa.c
  - Linux:
    >> mex CFLAGS="$CFLAGS -fopenmp" -lgomp mi_programa.c

5. Ficheros MEX y OpenMP

- Ejercicio 2. Paraleliza con OpenMP la función *prodmat*. Para ello:
  - Incorporamos el fichero de cabecera *omp.h*:
    #include <omp.h>
  - Para determinar el número de hilos, emplearemos la función *omp_set_num_threads(num_hilos)* fuera de la región paralela. Ejemplo para 4 hilos:
    omp_set_num_threads(4); /* Emplear 1, 2 o 4 hilos */
  - Usaremos la directiva *parallel for* para paralelizar el primero de los bucles, donde las variables *k* e *i* deberían ser privadas:
    #pragma omp parallel for private (k, i)

5. Ficheros MEX y OpenMP

- El código quedaría de esta manera:

```
#include <omp.h>
void prodmat (double *A, double *B, double *C, mwSize n) {
    mwSize i, j, k;
    omp_set_num_threads(4); /* Emplear 1, 2 o 4 hilos */
    #pragma omp parallel for private (k, i)
    for (j=0; j<n; j++)
        for (k=0; k<n; k++)
            for (i=0; i<n; i++)
                C[i+j*n]= C[i+j*n]+ A[i+k*n]* B[k+j*n];
}
```

21

5. Ficheros MEX y OpenMP

- Completa la siguiente tabla, a partir de los tiempos de ejecución obtenidos, empleando 1, 2 o 4 hilos, con matrices de tamaño 2000 x 2000:

	Fichero .m	Fichero mex 1 hilo	Fichero mex 2 hilos	Fichero mex 4 hilos
Tiempo (segs)				

22

5. Ficheros MEX y OpenMP

- Ejercicio 3. La función *diag* de Matlab tiene un comportamiento "curioso":
  - Si se le pasa como argumento de entrada una matriz, devuelve como resultado de salida un vector con la diagonal de la matriz:

```
>> A = [1 3; 5 7]; v = diag(A)
v =
     1
     7
```

  - Si se le pasa como dato de entrada un vector, devuelve como resultado una matriz diagonal cuyos elementos diagonales son los elementos del vector:

```
>> v = [1 7]; D = diag(v)
D =
     1   0
     0   7
```


5. Ficheros MEX y OpenMP

- Crea un fichero MEX, compuesto por una función en C llamada *diagonal*, que reproduzca el comportamiento de la función *diag*.

6. Depuración de archivos

- Para depurar archivos MEX en Windows necesitamos, por ejemplo, Visual Studio 2010, 2012, 2015, ..., 2019.
- Procedimiento:
  - Compila, desde Matlab, tu archivo MEX con la opción –g.
  - Sin cerrar Matlab, abre Visual Studio.
  - Abre el archivo .c que contiene el código y pon un punto de ruptura (pincha con el ratón en la línea apropiada).
  - En la opción “Depurar” del menú, selecciona la opción “Asociar a Proceso”. Busca el proceso Matlab y selecciónalo.
  - Ve a Matlab y ejecuta el programa MEX. Debe detenerse en el punto de ruptura de Visual Studio y, a partir de ahí, podrás depurarlo como cualquier otro programa en C.

6. Depuración de archivos

- En el siguiente enlace, se puede encontrar una guía para depurar archivos MEX en Linux, utilizando un depurador como *gdb*:

  https://es.mathworks.com/help/matlab/matlab_external/debugging-on-linux-platforms.html