Práctica 6. Programación con OpenMP

Métodos Numéricos para la Computación

Grado en Ingeniería Informática. Mención Computación
Escuela de Ingeniería Informática
Universidad de Las Palmas de Gran Canaria

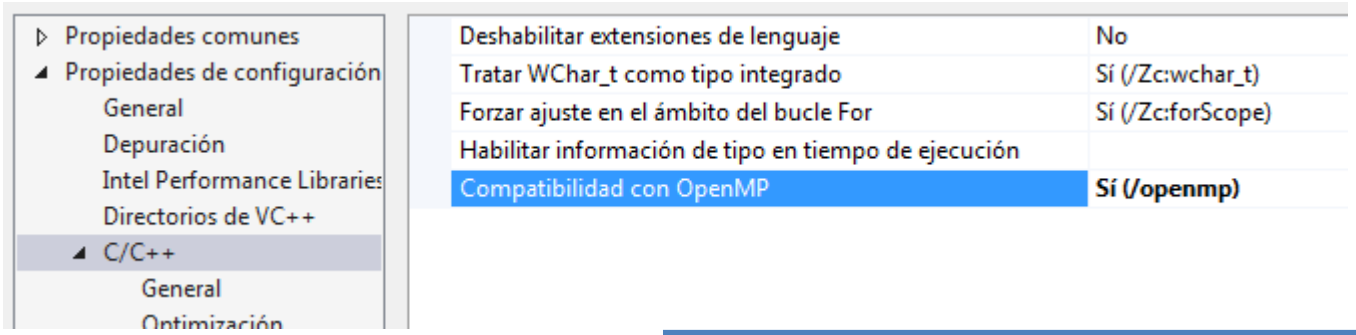
Contenidos

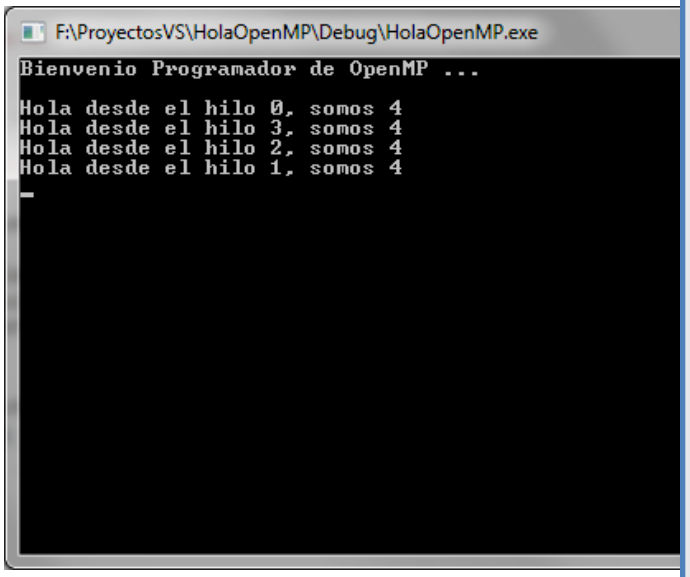
- Iniciación a la compilación de programas OpenMP
- Identificación
- Balanceo de Cargas
- Utilización conjunta de OpenMP y BLAS

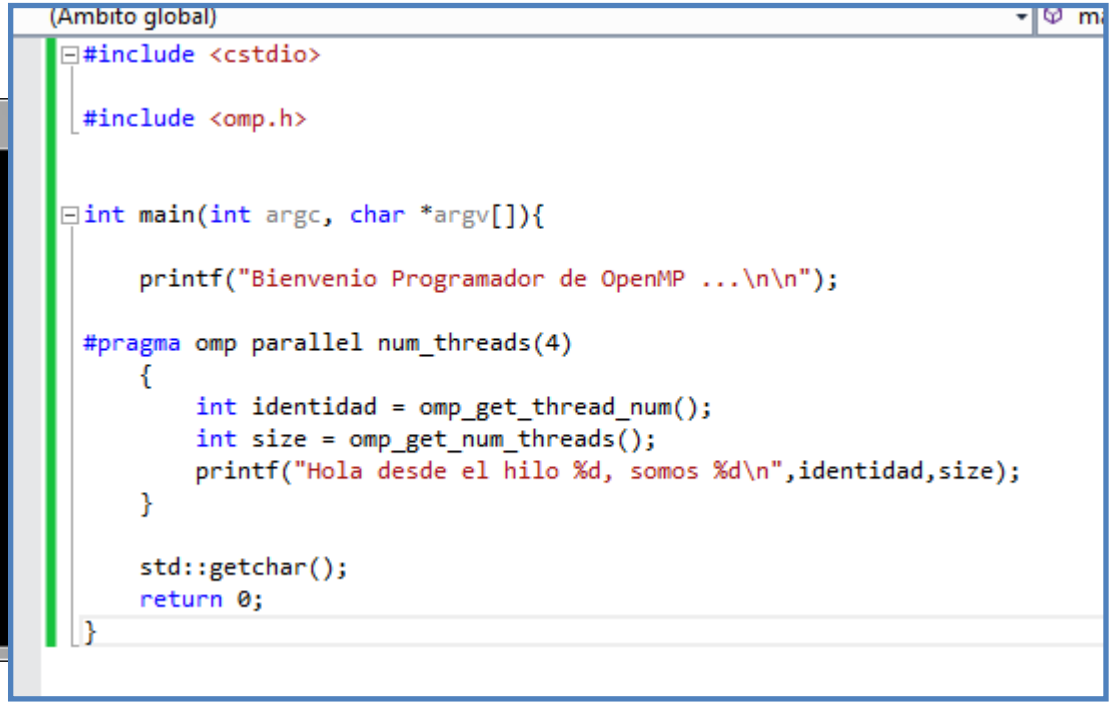
Tarea 1. Hola OpenMP

- Primer programa utilizando OpenMP, se utilizará identificación de hilo.
- Como utilizar OpenMP en Visual Studio

En Propiedades de configuración → C/C++ →Idioma activar Compatibilidad con OpenMP







El acceso a la pantalla con el printf() no es determinista y secuencial y por ello el orden de hilos en los mensajes no es predecible. Probar con un número superior

(Ámbito global) main(int argc, char * argv[])

```cpp
#include <cstdio>
#include <Windows.h>

#include <omp.h>



int main(int argc, char *argv[]){

    printf("Bienvenio Programador de OpenMP ...\n\n");

#pragma omp parallel num_threads(4)
    {
        int identidad = omp_get_thread_num();
        int size = omp_get_num_threads();
        printf("Hola desde el hilo %d, somos %d\n",identidad,size);
    }

    double inicio, fin, intervalo;
    inicio = omp_get_wtime();
    Sleep(1000); // stop 1000 milisegundos
    fin = omp_get_wtime();
    intervalo = omp_get_wtick();
    printf("\nTiempo transcurrido: %lf milisegundos \nprecision del tick: %g nanosegundos\n", (fin-inicio)*1.0e3, intervalo*1.0e9);

    std::getchar();
    return 0;
}
```

F:\ProyectosVS\HolaOpenMP\Debug\HolaOpenMP.exe

```
Bienvenio Programador de OpenMP ...

Hola desde el hilo 1, somos 4
Hola desde el hilo 0, somos 4
Hola desde el hilo 2, somos 4
Hola desde el hilo 3, somos 4

Tiempo transcurrido: 990.971025 milisegundos
precision del tick: 301.851 nanosegundos
```

Medición de tiempos aproximados. No es muy preciso. Error del orden del 1%

Tarea 2. OpenMP + BLAS

- En cada una de las operaciones individuales de los distintos hilos (que transcurren secuencialmente) se pueden utilizar por ejemplo las funciones de BLAS secuencial.
- Realizaremos la tarea de multiplicar una (gran) matriz descomponiéndola en trozos individuales que serán multiplicados por BLAS.
- El resultado será una multiplicación involucrando la totalidad de núcleos del sistema.

Fas e 1.Balanceo de Cargas

- Crear un nuevo proyecto, por ejemplo Balanceo, para probar un algoritmo para balancear las cargas de forma que el trabajo de los diversos hilos esté equilibrado.
- Añadir el fichero principal con main() y un fichero auxiliar con la función de balanceo de carga, por ejemplo BalanceoCarga.cpp, que usaremos posteriormente en otras tareas

```
F:\ProyectosVS\BigMult\Release\BigMult.exe

Tareas: 8, Hilos: 5
hilo:0 Numero:2 Posicion:0
hilo:1 Numero:2 Posicion:2
hilo:2 Numero:2 Posicion:4
hilo:3 Numero:1 Posicion:6
hilo:4 Numero:1 Posicion:7

Tareas: 3, Hilos: 7
hilo:0 Numero:1 Posicion:0
hilo:1 Numero:1 Posicion:1
hilo:2 Numero:1 Posicion:2
hilo:3 Numero:0 Posicion:3
hilo:4 Numero:0 Posicion:4
hilo:5 Numero:0 Posicion:5
hilo:6 Numero:0 Posicion:6

Tareas: 100, Hilos: 6
hilo:0 Numero:17 Posicion:0
hilo:1 Numero:17 Posicion:17
hilo:2 Numero:17 Posicion:34
hilo:3 Numero:17 Posicion:51
hilo:4 Numero:16 Posicion:68
hilo:5 Numero:16 Posicion:84
```

(Ámbito global)

```cpp
#include <cstdio>

#define MAXTHREADS 20

void BalanceoCarga(int Nth, int M, int *Pos, int *Num);

int main(int argc, char *argv[]){

    int Pos[MAXTHREADS], Num[MAXTHREADS];
    int Nth, M;

    Nth = 5;
    M = 8;
    printf("\nTareas: %d, Hilos: %d\n", M,Nth);
    BalanceoCarga(Nth, M, Pos, Num);
    for (int i = 0; i < Nth; i++){
        printf("hilo:%d Numero:%d Posicion:%d\n",i,Num[i],Pos[i]);
    }

    Nth = 7;
    M = 3;
    printf("\nTareas: %d, Hilos: %d\n", M, Nth);
    BalanceoCarga(Nth, M, Pos, Num);
    for (int i = 0; i < Nth; i++){
        printf("hilo:%d Numero:%d Posicion:%d\n", i, Num[i], Pos[i]);
    }

    Nth = 6;
    M = 100;
    printf("\nTareas: %d, Hilos: %d\n", M, Nth);
    BalanceoCarga(Nth, M, Pos, Num);
    for (int i = 0; i < Nth; i++){
        printf("hilo:%d Numero:%d Posicion:%d\n", i, Num[i], Pos[i]);
    }
```

Tres caso considerados:

1. Muy escaso en tareas. El balanceo resulta desequilibrado.
2. Tareas inferior a número de hilos. Igualmente desequilibrado, no se usa la totalidad de potencia.
3. Densamente poblado, mejor balanceado.

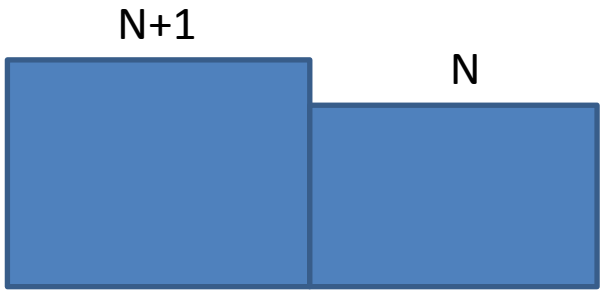
Asignación balanceada

(Ámbito global)

```
#include <cstdio>

void BalanceoCarga(int Nth, int M, int *Pos, int *Num){

    if (M <= Nth){
        for (int i = 0; i < Nth; i++) Pos[i] = i;
        for (int i = 0; i < M; i++) Num[i] = 1;
        for (int i = M; i < Nth; i++) Num[i] = 0;
        return;
    }
    int a = M / Nth;
    for (int i = 0; i < Nth; i++) Num[i] = a;
    int b = M - a*Nth;
    if (b > 0){
        for (int i = 0; i < b; i++) Num[i] ++;
    }
    Pos[0] = 0;
    for (int i = 1; i < Nth; i++) Pos[i] = Pos[i - 1] + Num[i - 1];
}
```

Asigna **M** tareas a **Nth** hilos, de forma lo más balanceada.
El resultado es el número de tareas **Num[i]** asignadas a cada hilo i, y la posición **Pos[i]** o índice da la primera tarea asignada a cada hilo.

Se asigna la parte entera de la división, y el resto se reparte entre los diversos hilos de uno en uno.



La diferencia de tareas por hilos nunca es mayor que la unidad. Para N grande, el resultado es un buen balanceo.

Fase 2. BigMult

- Crear un nuevo proyecto, por ejemplo BigMult, en el que experimentaremos la combinación de las dos tecnologías.
- Añadir el fichero principal con main() y una copia del fichero auxiliar BalanceoCarga.cpp,
- Activar ambos MKL y OpenMP.
- Probaremos inicialmente con matrices pequeñas para verificar el resultado comparándolo con MATLAB, posteriormente probaremos con grandes matrices.

Crear matrices de prueba

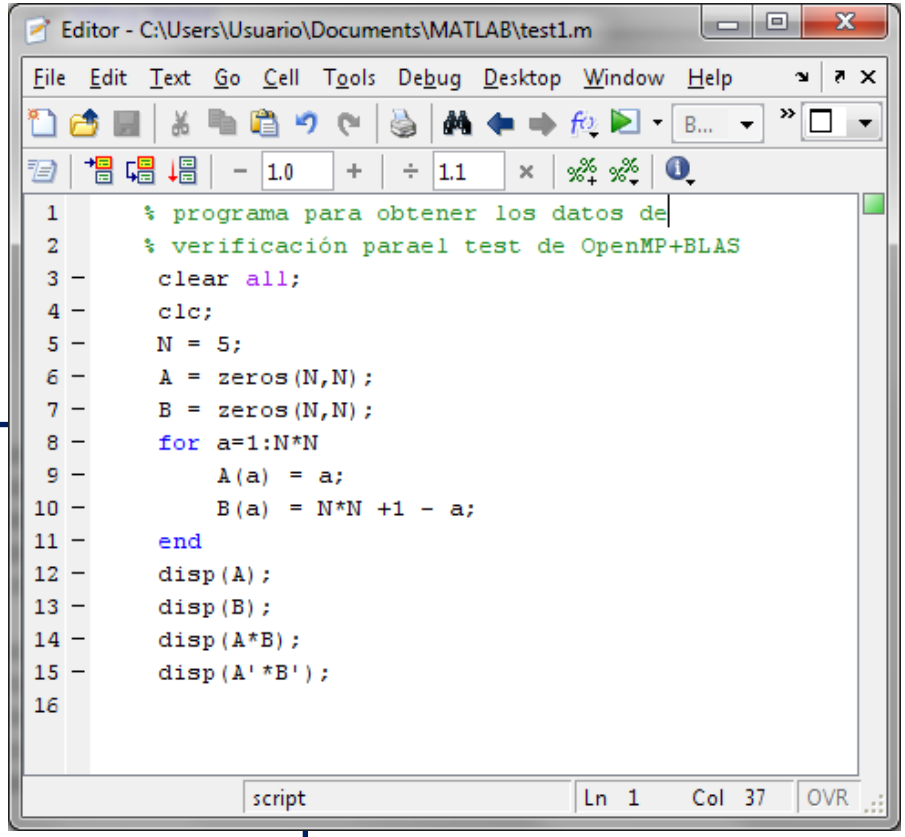
Crearemos en MATLAB dos matrices de prueba, A y B, y obtendremos el resultado de su multiplicación. Reproduciremos esas matrices en C/C++ y las multiplicaremos con utilización de BLAS y OpenMP. Verificaremos que el resultado es el correcto.

Las matrices se definen como espacios lineales de 25 número:

$$A = (1, 2, 3, \ldots, 24, 25)$$
$$B = (25, 24, \ldots, 3, 2, 1)$$

Se interpreta como una matriz de 5x5 por filas como:

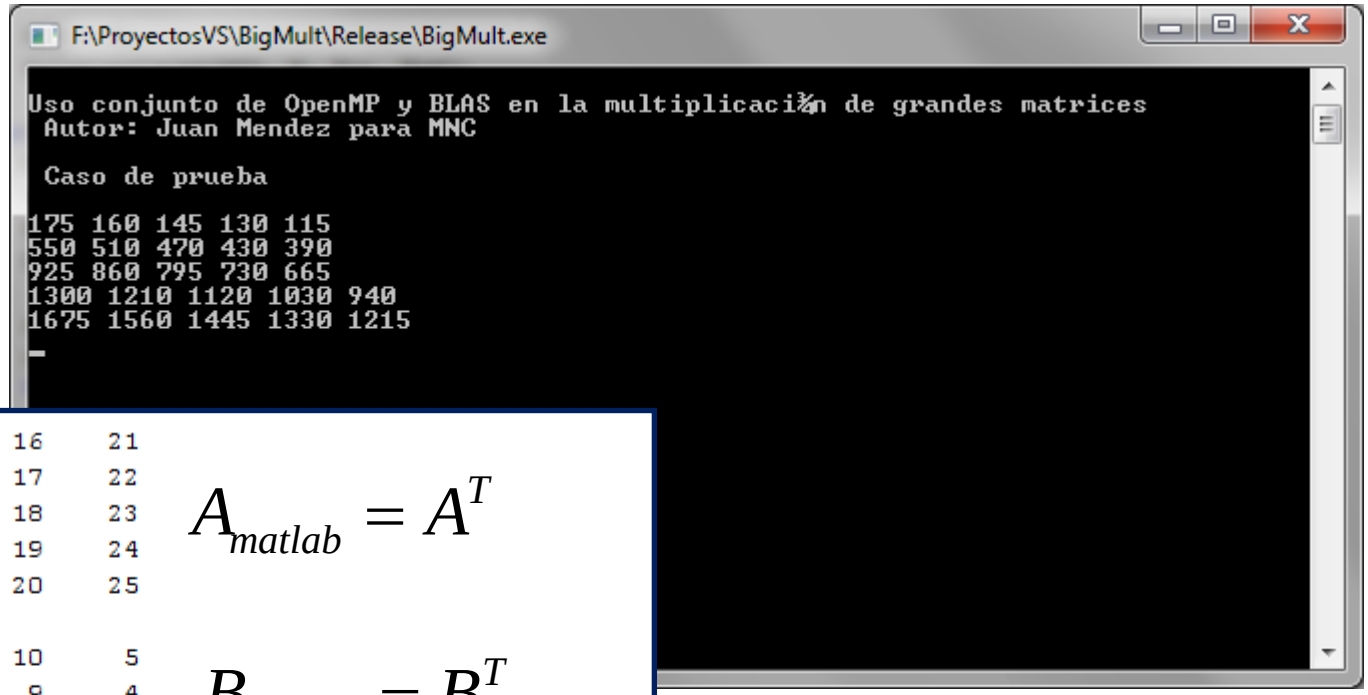
$$A = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{pmatrix}$$




```
     1     6    11    16    21
     2     7    12    17    22
     3     8    13    18    23
     4     9    14    19    24
     5    10    15    20    25

    25    20    15    10     5
    24    19    14     9     4
    23    18    13     8     3
    22    17    12     7     2
    21    16    11     6     1

    1215    940    665    390    115
    1330   1030    730    430    130
    1445   1120    795    470    145
    1560   1210    860    510    160
    1675   1300    925    550    175

     175    160    145    130    115
     550    510    470    430    390
     925    860    795    730    665
    1300   1210   1120   1030    940
    1675   1560   1445   1330   1215
```

MATLAB codifica por columnas, luego el producto A*B no es el que resultará si en BLAS codificamos con filas.

Si será el traspuesto: A'*B'




```
 1     6    11    16    21
 2     7    12    17    22
 3     8    13    18    23
 4     9    14    19    24
 5    10    15    20    25

25    20    15    10     5
24    19    14     9     4
23    18    13     8     3
22    17    12     7     2
21    16    11     6     1

    1215      940      665      390      115
    1330     1030      730      430      130
    1445     1120      795      470      145
    1560     1210      860      510      160
    1675     1300      925      550      175

     175      160      145      130      115
     550      510      470      430      390
     925      860      795      730      665
    1300     1210     1120     1030      940
    1675     1560     1445     1330     1215
```

$$A_{matlab} = A^T$$

$$B_{matlab} = B^T$$

$$A_{matlab} B_{matlab} = A^T B^T$$

$$A_{matlab}{}^T B_{matlab}{}^T = AB$$

Declaraciones

Para comprobar el procedimiento, pudiendo aprovechar el máximo de capacidad de computo, realizaremos un producto de dos matrices aleatoria de 4000x4000. EL caso de prueba lo realizaremos con las matrices de 5x5 antes definidas. El máximo número de hilos que podremos usar es de 20. En la practica con procesadores i7 el máximo de potencia se obtendrá alrededor de 4 hilos

```
#include <cstdio>
#include <random>

#include <mkl.h>
#include <omp.h>

#define Nbig 4000
#define MAXTHREADS 20

extern void BalanceoCarga(int Nth, int M, int *Pos, int *Num);

int main(int argc, char *argv[]){

    printf("\nUso conjunto de OpenMP y BLAS en la multiplicación de grandes matrices\n Autor: Juan Mendez para MNC\n");

    int Pos[MAXTHREADS], Num[MAXTHREADS];
    int Nth, N;
    double *A, *B, *C;
```

Test o computación intensiva

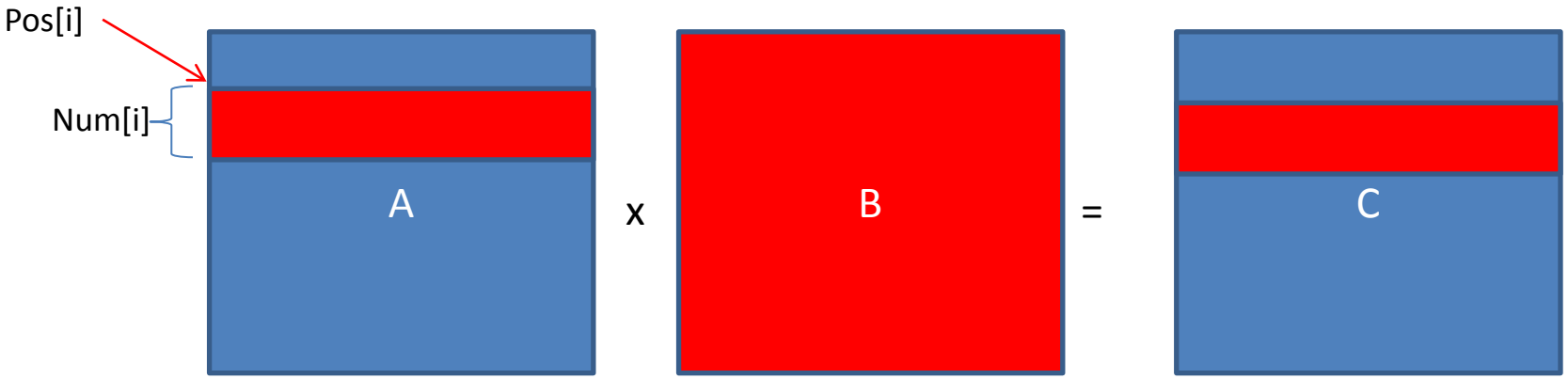
```
bool test = true; // true para verificar el funcionamiento, false con big
if (test){
    printf("\n Caso de prueba\n\n");
    N = 5;
    A = (double *)mkl_malloc(N*N*sizeof(double), 64);
    B = (double *)mkl_malloc(N*N*sizeof(double), 64);
    C = (double *)mkl_malloc(N*N*sizeof(double), 64);

    for (int i = 0; i < N*N; i++){
        A[i] = 1.0 + (double)i;
        B[i] = (double)(N*N) - (double)i;
    }

    Nth = 3;
    BalanceoCarga(Nth, N, Pos, Num);
}
else{
    N = Nbig;
    A = (double *)mkl_malloc(N*N*sizeof(double), 64);
    B = (double *)mkl_malloc(N*N*sizeof(double), 64);
    C = (double *)mkl_malloc(N*N*sizeof(double), 64);

    std::default_random_engine generador;
    std::normal_distribution<double> distribucion(0.0, 1.0);
    for (int i = 0; i < N*N; i++){
        A[i] = distribucion(generador);
        B[i] = distribucion(generador);
    }

    Nth = 2;   // deberemos utilizar diversos valores
    BalanceoCarga(Nth, N, Pos, Num);
}
```

Rellenaremos las matrices acorde al tipo de pruebas que realicemos.

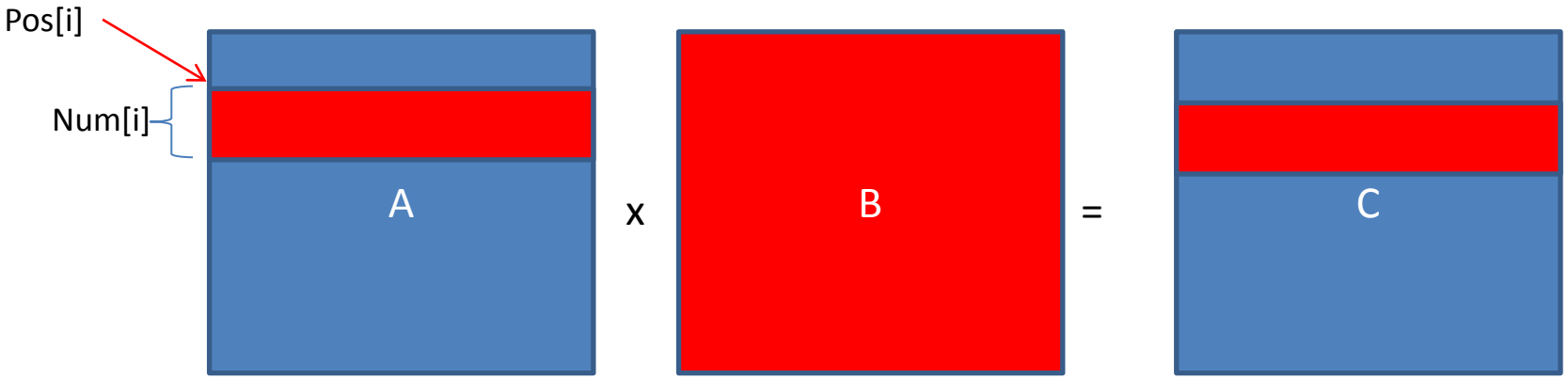
En el caso de prueba usaremos 3 hilos para multiplicar matrices de 5x5 para verificar que la distribución paralela, aunque es evidentemente ineficiente, no influye en la obtención del resultado correcto

Descomposición en Bloques. Asignaremos grupos de filas completas a cada hilo, de forma que un bloque de resultados en C se obtiene mediante la multiplicación de un bloque equivalente en A y toda la matriz B.

El balanceo de carga correcto determinará cuantas líneas forman cada bloque y donde comienza ese bloque.



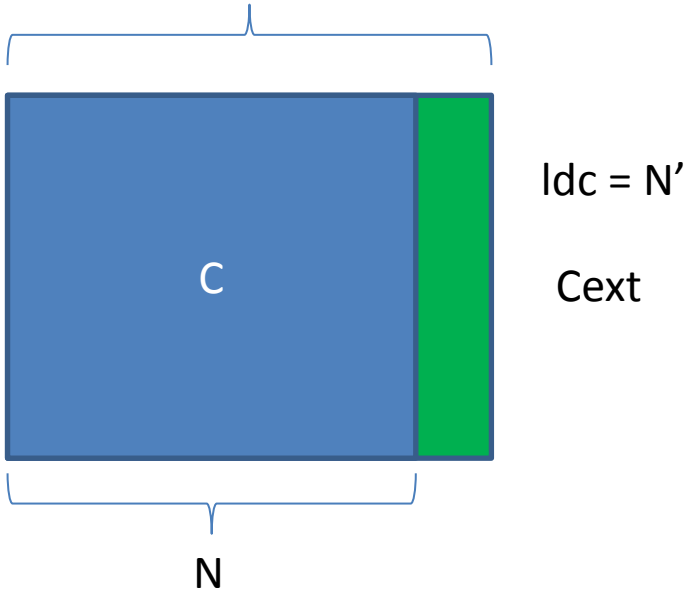
```
    // fork
    int i; // el indice del for no puede ser dinamico
    double inicio = omp_get_wtime(); // utilizamos el reloj de OpenMP
#pragma omp parallel for private(i) num_threads(Nth)
    for (i = 0; i < Nth; i++){
        //printf("Hilo: %d, Desde: %d, Nlineas: %d\n",i,Pos[i],Num[i]); // solamente en las pruebas
        cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, Num[i], N, N, 1.0, &(A[Pos[i]*N]), N, B, N, 0.0, &(C[Pos[i]*N]), N);
    }
    double fin = omp_get_wtime();
```



La matriz B es utilizada por todos los hilos, pero solamente en lectura. La matriz C es escrita por todos los hilos, pero en filas diferentes. Pudiera existir colisión dentro de una página de memoria, que no será muy frecuente, pero no colisión en filas.

Evitar colisiones de página

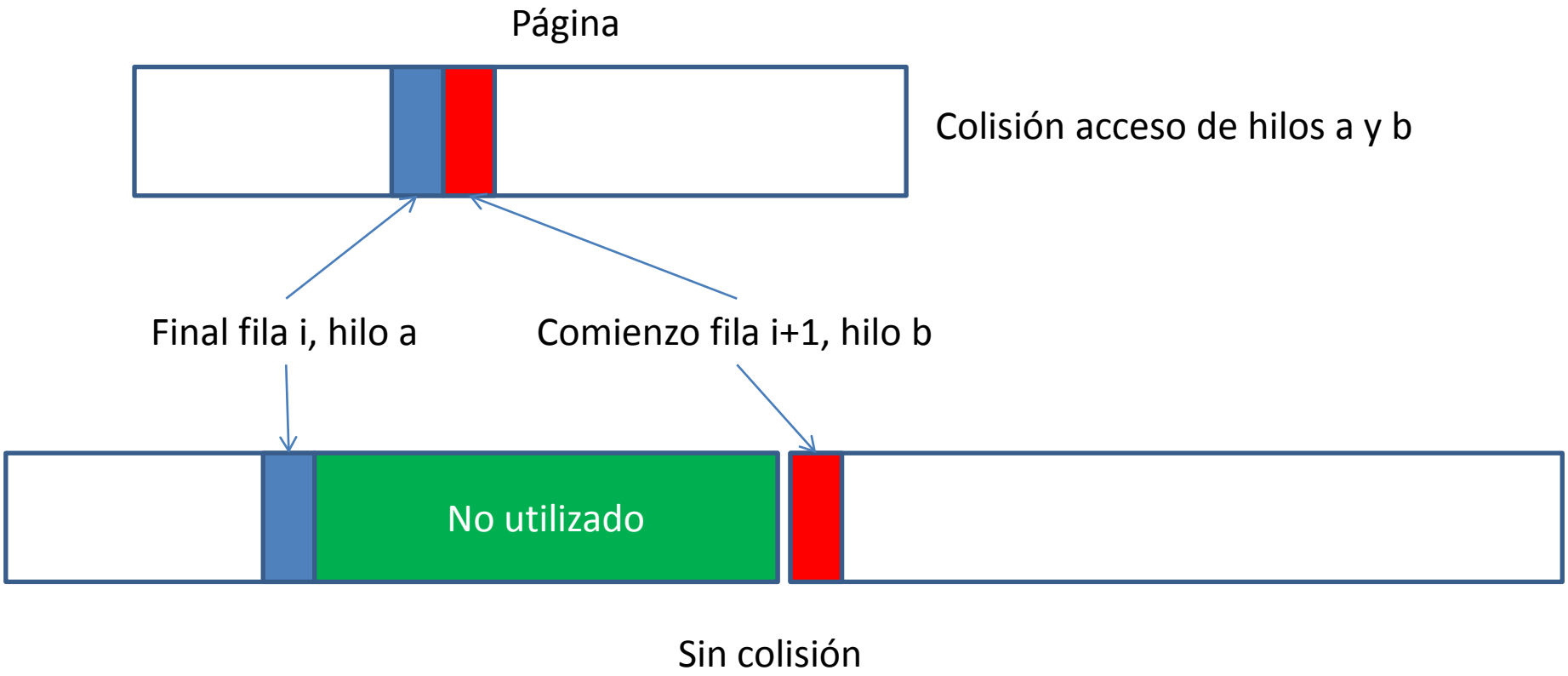
N'*sizeof(double) múltiplo de _SC_PAGESIZE



El tamaño que es necesario reservar en memoria para cada matriz es NxN'

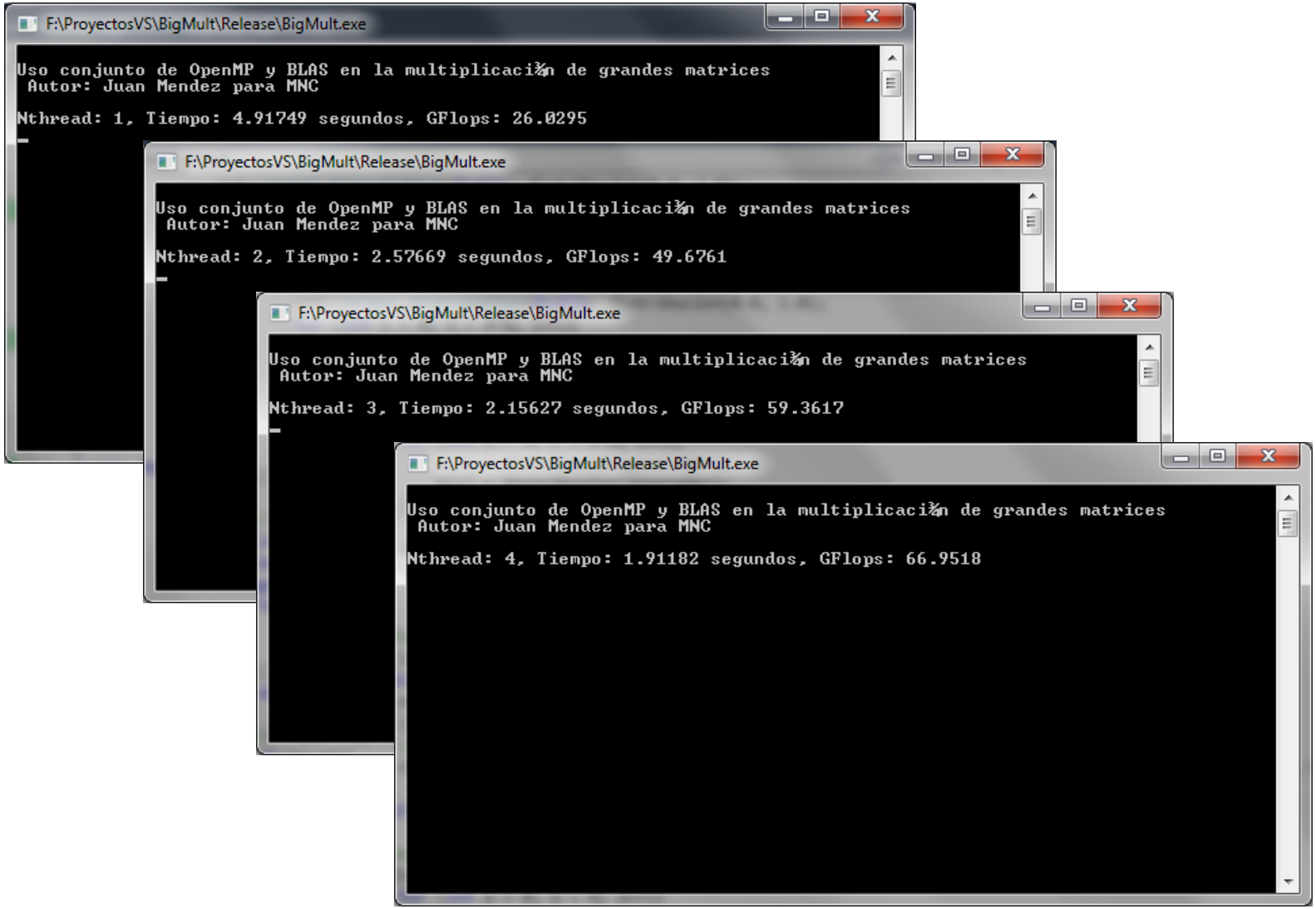
Codificar las matrices A,B y C dentro de una extendida, donde el ancho de las columnas es múltiplo del tamaño de la página de memoria en cache. Posteriormente utilizar N para indicar el número de columnas, pero N' para lda, ldb y ldc. De esta forma el final de una fila nunca estará en la misma pagina de cache que el comienzo de la fila siguiente y los hilos diferentes no tratará de acceder a la misma página de memoria central en cache.

Evitar colisiones de página



Reporte de resultados

```
    // fork
    int i; // el indice del for no puede ser dinamico
    double inicio = omp_get_wtime(); // utilizamos el reloj de OpenMP
#pragma omp parallel for private(i) num_threads(Nth)
    for (i = 0; i < Nth; i++){
        //printf("Hilo: %d, Desde: %d, Nlineas: %d\n",i,Pos[i],Num[i]); // solamente en las pruebas
        cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans, Num[i], N, N, 1.0, &(A[Pos[i]*N]), N, B, N, 0.0, &(C[Pos[i]*N]), N);
    }
    double fin = omp_get_wtime();

    // report de resultados
    if (test){
        for (int a = 0; a < N; a++){
            for (int b = 0; b < N; b++) printf("%g ",C[a*N+b]);
            printf("\n");
        }
    }
    else{
        double tiempo = fin - inicio;
        double Gflops = 2.0*N*N*N*1.0e-09/tiempo;
        printf("\nNthread: %d, Tiempo: %g segundos, GFlops: %g\n",Nth,tiempo,Gflops);
    }
    mkl_free(A);
    mkl_free(B);
    std::getchar();
    return 0;
}
```



¿Y más allá de 4 hilos?

Tareas específicas

- Construir el programa y verifica el test con la matriz de 5x5
- Ejecutar el programa para valores del número de hilos entre 1 y 10. Construir una tabla de tiempos y Gflops.
- Obtener el Speedup para cada número de hilos utilizado.

$$S(n) = \frac{T(1)}{T(n)}$$

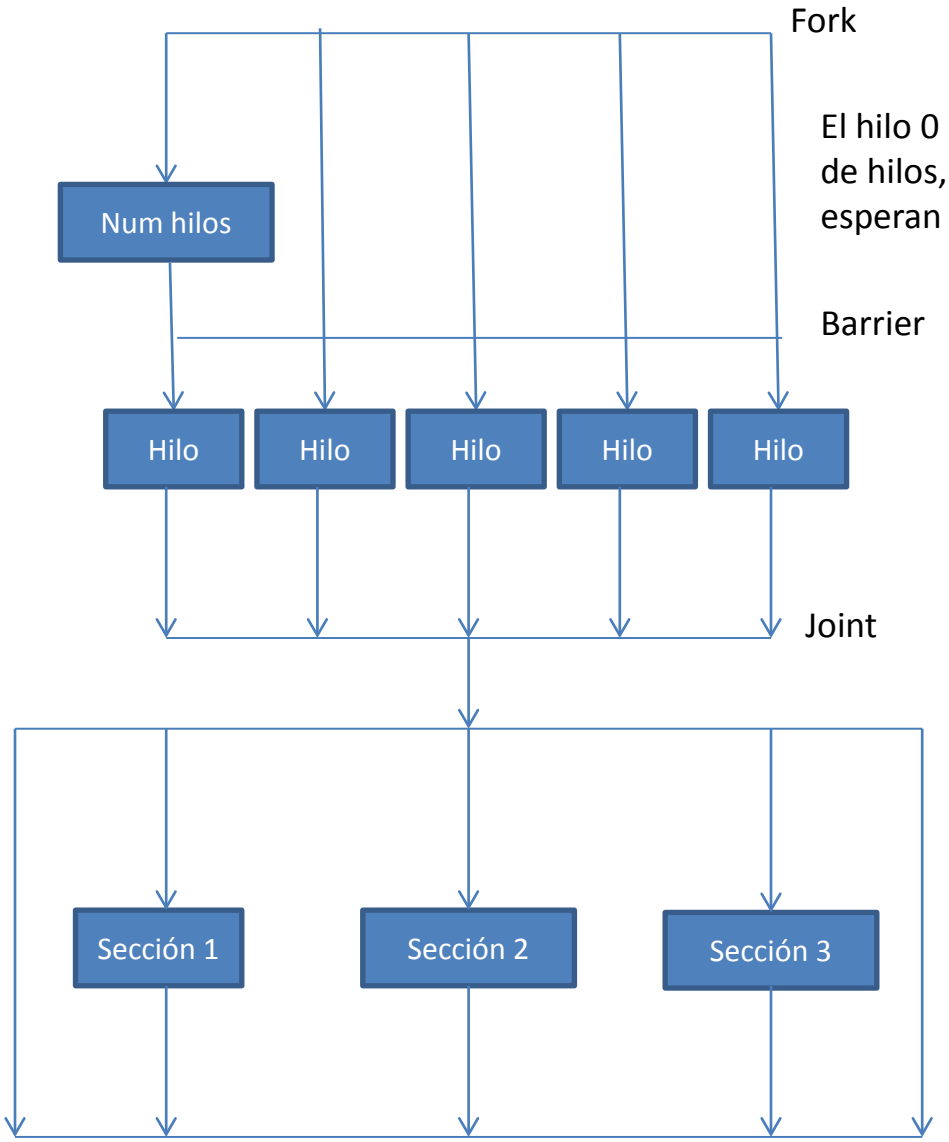
- Si se dispone del tiempo serial de prácticas anteriores, computar también;

$$S_{serial}(n) = \frac{T_{serial}}{T(n)}$$

- Construir una gráfica con la relación Speedup vs el número de hilos
- Construir una gráfica con la potencia en Gflops en función del número de hilos.
- Discutir los resultados.

Tarea 3. Uso de barrier y sections

- Ejemplo que ilustra el uso de una barrera para sincronizar diversas hilos.
- Se ilustra igualmente el uso de secciones. Cada section de una directiva sections es ejecutada por un hilo distinto. Si el número de hilos es mayor que el de section, entonces los restantes hilos no ejecutan tareas.



Esquema de la Tarea de Barrera y Secciones

Ejemplo parcial con tres secciones y 10 hilos.

```
G:\ProyectosVS\openmpSections\Release\openmpSections.exe
Numero de hilos: 10
Hilo 8
Hilo 3
Hilo 2
Hilo 1
Hilo 0
Hilo 5
Hilo 4
Hilo 9
Hilo 6
Hilo 7
Hilo 0 pasa por la seccion 1
Hilo 2 pasa por la seccion 2
Hilo 4 pasa por la seccion 3
```

```
#pragma omp parallel num_threads(Nth) // fork
{
        int tid = omp_get_thread_num();
        if(tid==0) printf("Numero de hilos: %d\n", omp_get_num_threads());
        #pragma omp barrier
        printf("Hilo %d\n",tid);
}

#pragma omp parallel sections num_threads(Nth) // fork
{
        #pragma omp section
        {
            int tid = omp_get_thread_num();
            printf("Hilo %d pasa por la seccion 1\n",tid);
        }
        #pragma omp section
        {
            int tid = omp_get_thread_num();
            printf("Hilo %d pasa por la seccion 2\n",tid);
        }
        #pragma omp section
        {
            int tid = omp_get_thread_num();
            printf("Hilo %d pasa por la seccion 3\n",tid);
        }
}  // Joint
```

¿Qué hemos aprendido?

- Como informar al compilador de Visual Studio para que acepte las directivas de OpenMP
- Utilizar algunas funciones omp_
- Lanzar un fork con parallel y parallel for
- Descomponer un problema matricial en trozos más pequeños para poder utilizar paralelismo.
- Balancear cargas en casos sencillos.
- Hacer cooperar las tecnologías BLAS y OpenMP para incrementar la capacidad de computo de sistema de múltiples núcleos.
- Valorar la influencia del número de hilos utilizados en función del número de núcleos del sistema.
- Utilizar barreras y secciones.

Qué debe entregar el alumno

- Cada alumno entregará en el Campus Virtual una memoria en PDF o Word en la que estará contenida una descripción del trabajo realizado, incluyendo descripción, el listado MATLAB o C de la actividad realizada y la captura de pantalla de las gráficas o imágenes generadas. Para autentificar las imágenes cuando sea posible el alumno incluirá su nombre en cada imagen mediante la función title().
- En principio la tarea quedará abierta para su entrega hasta cierta fecha que se indicará.
- Se puede trabajar en grupo en el Laboratorio, pero la memoria elaborada y entregada será individual.

Bibliografía

- (Barn11a) B. Barney, OpenMP, Lawrence Livermore National Laboratory, 2011 https://computing.llnl.gov/tutorials/openMP/
- (Chan01) R. Chandra, L. Dagum, D. Kohr, D. Maydan, J. McDonal y R. Menon, Parallel Programming in OpenMP, , Academic Press, 2001.
- (Chap08) B. Chapman, G. Jost y R. van der Pas, Using OpenMP: Portable Shared Memory Parallel Programming, MIT University Press, 2008.
- (Qinn03) M. J. Qinn, Parallel Programming in C with MPI and OpenMP, MacGraw-Hill, 2003.
- OpenMP organización. http://openmp.org/wp/