Laboratorio de Paralelismo

OpenMP: ejemplos y ejercicios

> Máquinas y directorios

Vamos a trabajar con el siguiente multiprocesador SMP:

PowerEdge 6850 (DELL)
4 procesadores Intel Xeon 7100 de doble núcleo a 2.6 GHz
1 GB de RAM
Conexión Gigabit Ethernet

La dirección de la máquina es `acpt51.gi.ehu.es` Se puede acceder desde cualquier punto de la facultad y desde fuera de la red de la universidad (por ejemplo, desde casa) a través de VPN.

En la cuenta de cada grupo hemos creado un directorio, `LPAR`, con unos cuantos programas de ejemplo y ejercicios. Cuando vayas a trabajar con ellos, haz una copia a otra carpeta.
OJO: no trabajes en esa carpeta ni dejes ahí nada que necesites, porque es posible que la modifiquemos.

> Compilación y ejecución de programas

Para compilar los programas, vamos a utilizar el compilador de C/C++ de Intel (si quieres, lo puedes bajar desde la página de Intel e instalar en tu ordenador).

Para compilar un programa de nombre `prog.c`, que incluya directivas y funciones de OpenMP, basta con ejecutar:

```
> icc –openmp [-Ox] –o prog prog.c
```
(sin `–openmp`, versión serie)

Si estamos interesados en el tiempo de ejecución, compilaremos con la optimización `–O2` tanto en serie como en paralelo (está puesta por defecto y utiliza las optimizaciones más habituales: registros, *loop unrolling*, rutinas *in-line*…). Las otras opciones principales son `–O0` (genera un código muy poco eficiente) y `–O3` (realiza una optimización más agresiva que puede no dar mejores resultados, se puede probar para cada programa y ver si merece la pena).

Si no hay errores, se habrá creado el correspondiente ejecutable. Recuerda que el número de hilos a utilizar se puede controlar, entre otras maneras, mediante una variable de entorno:

```
> export OMP_NUM_THREADS=xx
```
xx = número de *threads*

```
/******************************************************************************
  hola1.c
  Generacion de varios threads y modificacion del numero de threads mediante:
     export OMP_NUM_THREADS=4
******************************************************************************/

#include <stdio.h>

#ifdef _OPENMP
  #include <omp.h>
  #define TRUE  1
  #define FALSE 0
#else
  #define omp_get_thread_num()  0
  #define omp_get_num_threads() 1
#endif

#define N  24

int     tid, nthr;              // identificador del thread y numero de threads
int     i, A[N];

main  ()
{
  for (i=0; i<N; i++) A[i]=0;

  #pragma omp parallel private(tid)
  {
    nthr = omp_get_num_threads();
    tid = omp_get_thread_num();
    printf ("Thread %d de %d en marcha \n", tid, nthr);

    A[tid] = tid + 10;
    printf ("El thread %d ha terminado \n", tid);
  }

  for (i=0; i<N; i++) printf ("A(%d) = %d \n", i, A[i]);
}
```

```
/******************************************************************************
  hola2.c
  Definición del numero de threads mediante la funcion  omp_set_num_threads()
******************************************************************************/

...
main  ()
{
  printf("\nIntroduce el numero de threads  --->  ");
  scanf("%d",&nthr);

#ifdef _OPENMP
  omp_set_num_threads(nthr);
#endif

  #pragma omp parallel private(tid)
  {
    tid = omp_get_thread_num();
    printf ("Thread %d en marcha \n", tid);

    A[tid] = tid + 10;
    printf ("El thread %d ha terminado \n", tid);
  }

  for (i=0; i<N; i++) printf ("A(%d) = %d \n", i, A[i]);
}
```

```
/******************************************************************************
  hola3.c
  Los tres casos juntos (pej: export con 16, teclado con 8, funcion con 6 threads)
******************************************************************************/

...

main  ()
{
// el numero de hilos se controla mediante la variable de entorno

  #pragma omp parallel private(tid)
  {
    tid = omp_get_thread_num();
    printf ("Thread %d en marcha \n", tid);

    A[tid] = tid + 10;
    printf ("El thread %d ha terminado \n", tid);
  }

  for (i=0; i<N; i++) printf ("A(%d) = %d \n", i, A[i]);
  printf("\n\n fin de la primera region paralela \n\n");


// el numero de hilos se controla mediante una funcion

  printf("\n\nIntroduce el numero de threads  --->  ");
  scanf("%d",&nthr);

#ifdef _OPENMP
  omp_set_num_threads(nthr);
#endif

  #pragma omp parallel private(tid)
  {
    tid = omp_get_thread_num();
    printf ("Thread %d en marcha \n", tid);

    A[tid] = tid + 100;
    printf ("El thread %d ha terminado \n", tid);
  }

  for (i=0; i<N; i++) printf ("A(%d) = %d \n", i, A[i]);
  printf("\n\n fin de la segunda region paralela \n\n");



// el numero de hilos se controla mediante una clausula
  #pragma omp parallel private(tid) num_threads(6)
  {
    tid = omp_get_thread_num();
    printf ("Thread %d en marcha \n", tid);

    A[tid] = tid + 1000;
    printf ("El thread %d ha terminado \n", tid);
  }

  for (i=0; i<N; i++) printf ("A(%d) = %d \n", i, A[i]);
  printf("\n\n fin de la tercera region paralela \n\n");
}
```

```
/******************************************************************************
 var1.c
 ambito de variables (ejemplo transparencias )
******************************************************************************/
//Predecir antes de ejecutar lo que tiene que salir
//Ojo con el uso de variables private sin inicializar (-> firstprivate)

#include <stdio.h>

#ifdef _OPENMP
  #include <omp.h>
  #define TRUE  1
  #define FALSE 0
#else
  #define omp_get_thread_num()  0
  #define omp_get_num_threads() 1
#endif

int tid, x=-1, y=-1, z=-1;

main ()
{
  x = 2;
  y = 1;

  printf("\n\n    ANTES     --> x = %d\t y = %d\t z = %d\n", x, y, z);

  #pragma omp parallel shared(y) private(x,z)         // probar con x firstprivate
  {
    tid = omp_get_thread_num();

   if (tid==0) printf("\n\n    DENTRO(1) --> x = %d\t y = %d\t z = %d\n", x, y, z);

    z = x*x + 3;
    x = y*3 + z;
    if (tid==0) y = x;

    if (tid==0) printf("\n\n     DENTRO(2) --> x = %d\t y = %d\t z = %d\n", x, y,
z);
  }

  printf("\n\n    FUERA     --> x = %d\t y = %d\t z = %d\n\n", x, y, z);
}
```

```
/******************************************************************************
 var2.c
 hay que declarar correctamente las variables. PARA COMPLETAR.
******************************************************************************/
//// Ejecutar var_g para que se vea que tiene que salir

#include <stdio.h>
#include <omp.h>

#ifdef _OPENMP
  #include <omp.h>
  #define TRUE  1
  #define FALSE 0
#else
  #define omp_get_thread_num()  0
  #define omp_get_num_threads() 1
#endif
#define N 4

int tid, nth;
int comienzo, fin, x = 0, z = 0, zpar, A[N][N], i, j;

main ()
{
  for (i=0; i<N; i++)
   for (j=0; j<N; j++) A[i][j] = i+j;

  #pragma omp parallel default(none)
  {
    tid = omp_get_thread_num();
    nth = omp_get_num_threads();

    comienzo = tid * N / nth;
    fin = (tid+1) * N / nth;

    printf("\n  Proceso %d/%d; comienzo = %d, fin = %d \n", tid, nth, comienzo, fin);

    zpar = 0;

    for (i=comienzo; i<fin; i++)
    for (j=0; j<N; j++)
    {
      x = A[i][j] * A[i][j];
      A[i][j] = A[i][j] + x;
      zpar = zpar + x;
    }
    z = z + zpar;
  }

  printf("\n    --> Matriz A[i][j]\n");
  for(i=0; i<4; i++)
  {
    printf("\n");
    for(j=0; j<4; j++) printf(" %8d ", A[i][j]);
    printf("\n");
  }
  printf("\n\n    --> x, z, zpar =  %d, %d, %d \n\n", x, z, zpar);
}
```

```
/******************************************************************************
  threadp.c   OMP
  Ejemplo de uso de variables threadprivate
  RP1: copyin(x) --> x toma el valor inicial del master; y/z sin inicializar (0)
  RP2: x/y mantienen el valor de la region paralela anterior (threadprivate)
  RP3: una rutina: x/z mantienen valor, pero z no es privada (var. global original)
       tid si es privada porque se pasa como parametro.
******************************************************************************/
#include <stdio.h>

#ifdef _OPENMP
  #include <omp.h>
  #define TRUE  1
  #define FALSE 0
#else
  #define omp_get_thread_num()  0
  #define omp_get_num_threads() 1
#endif

int     x, y, z, i, tid;
#pragma omp threadprivate (x, y)

void imprimir(int tid)
{
  printf("     Thread %d:  x = %d   y = %d   z = %d \n", tid, x, y, z);
}

main  ()
{
  x = y = z = 5;
  printf("\n\n  Valores antes de la region paralela:  x, y, z = %d %d %d \n", x, y, z);

  printf("\n  Region paralela 1:  x/y threadprivate; x copyin; z/tid private\n\n");
  #pragma omp parallel private(z,tid) copyin(x)
  {
    tid = omp_get_thread_num();
    printf("  A: Thread %d:  x = %d   y = %d   z = %d \n", tid, x, y, z);

    x = x + tid + 1;
    y =  tid;
    z =  tid + 20;

    #pragma omp barrier

    printf("  B: Thread %d:  x = %d   y = %d   z = %d \n", tid, x, y, z);
  }

  printf(" \n   >>>>>>>> Ejecucion de un trozo en serie \n ");

  printf("\n  Region paralela 2:  x/y threadprivate; z/tid private\n\n");
  #pragma omp parallel private(z,tid)
  {
    tid = omp_get_thread_num();
    printf("     Thread %d:  x = %d   y = %d   z = %d \n", tid, x, y, z);
  }

  printf(" \n   >>>>>>>> Ejecucion de un trozo en serie \n ");

  printf("\n  Region paralela 3:  se imprime desde una rutina: tid como parametro \n\n");
  #pragma omp parallel private(z,tid)
  {
    tid = omp_get_thread_num();
    imprimir(tid);
  }

  printf("\n  Main:  se imprime desde una rutina: -1 como parametro \n\n");
  imprimir(-1);

  printf("\n  Fin del main:  x = %d   y = %d   z = %d \n\n", x, y, z);
}
```

```
/******************************************************************************
   repar.c
   reparto de interacciones de un bucle entre los procesos.
   hacer un reparto consecutivo del primer bucle y entrelazado del segundo
      PARA COMPLETAR
******************************************************************************/
//// Ejecutar repar_g para que se vea la salida del programa

#include <stdio.h>

#ifdef _OPENMP
  #include <omp.h>
  #define TRUE  1
  #define FALSE 0
#else
  #define omp_get_thread_num()  0
  #define omp_get_num_threads() 1
#endif
#define N 40

main ()
{
  int   A[N], B[N];
  int   i, j;

// ejecutar los bucles en paralelo haciendo el reparto de iteraciones a mano:
// (bucle 1) consecutivo y (bucle 2) entrelazado
// para ver que thread ejecuta cada iteracion, cargar en A[i] y B[i] el tid del proceso
// en lugar de i



  for (i=0; i<N; i++) A[i] = i;
  for (i=0; i<N; i++) B[i] = i;



//  impresion de resultados

  printf ("\n      Vector A, reparto consecutivo  \n\n");
  printf ("  0   1   2   3   4   5   6   7   8   9\n");
  printf ("----------------------------------------\n");
  for (i=0; i<N/10; i++)
  {
    printf ("\n");
    for (j=i*10; j<(i+1)*10; j++) printf("%3d ", A[j]);
    printf ("\n");
  }
  printf ("\n\n");
  printf ("      Vector B, reparto entrelazado  \n\n");
  printf ("  0   1   2   3   4   5   6   7   8   9\n");
  printf ("----------------------------------------\n");
  for (i=0; i<N/10; i++)
  {
    printf ("\n");
    for (j=i*10; j<(i+1)*10; j++) printf("%3d ", B[j]);
    printf ("\n");
  }
  printf ("\n");
}
```

```
/******************************************************************************
   sched.c
   Ejemplo para analizar la planificacion o scheduling
   la planificacion se indica en una variable de entorno (Runtime)
    ejemplo:    export OMP_SCHEDULE="static,4"
******************************************************************************/

#include <omp.h>
#include <stdio.h>
#include <unistd.h>

#define N 40

main ()
{
  int   tid;
  int   A[N];
  int   i;

  for(i=0; i<N; i++) A[i]=-1;

#pragma omp parallel for schedule(runtime) private(tid)
  for (i=0; i<N; i++)
  {
    tid = omp_get_thread_num();
    A[i] = tid;
    usleep(1);
  }

  for (i=0; i<N/2; i++)  printf (" %2d", i);
  printf ("\n");
  for (i=0; i<N/2; i++)  printf (" %2d", A[i]);
  printf ("\n\n\n");
  for (i=N/2; i<N; i++)  printf (" %2d", i);
  printf ("\n");
  for (i=N/2; i<N; i++)  printf (" %2d", A[i]);
  printf ("\n\n");
}
```

```
/******************************************************************************
  tarea.c
  Ejemplo para ver el efecto del reparto en el tiempo de ejecucion
  Ejecutar modificando el scheduling: static,static 4, dynamic
******************************************************************************/

#include <omp.h>
#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>

struct timeval  t0, t1;
double          tej;

#define N 4000

double calculo (int veces)
{
  usleep(veces);
  return(1);
}

int    i, A[N], nth=-1;
double total;

main ()
{
//Inicializacion vector de tamano de tareas
  for (i=0; i<N; i++) A[i] = 1;

  A[1] = 100000;
  A[5] = 100000;

  gettimeofday(&t0, 0);

  total = 0.0;

 #pragma omp parallel for schedule(runtime) reduction (+:total)
  for (i=0; i<N; i++)
    total +=  calculo(A[i]);

  gettimeofday(&t1, 0);
  tej = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec - t0.tv_usec) / 1e6;

  #ifdef _OPENMP
    nth = omp_get_max_threads();
  #endif

  printf("\n\n Tej con %d hilos (-1=serie) = %1.3f s\n\n Total= %.2f\n\n", nth, tej, total);
}
```

Ejercicio: `matvec.c`

Escribir un programa que efectúe el siguiente cálculo con matrices y vectores:

```
double A(NxN), B(N), C(N), D(N), X

C(N) = A(NxN) × B(N)
D(N) = A(NxN) × C(N)
X = C(N) . D(N)
```

El programa debe pedir al principio el tamaño de los vectores, `N` (máximo, `N` = 1000) y el número de *threads*.

```
/******************************************************************************
   sec.c
   Ejemplo de secciones en paralelo (ver orf.c para directivas huerfanas)
******************************************************************************/

//// Probar con 4 threads, con 3 (lo mismo) y con 2 (el doble de tiempo)

#include <omp.h>
#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>

#define NA 2000000
#define NB 2000000
#define NC 2000000

struct timeval  t0, t1;
double          tej;

double func (int N)
{
  usleep(N);
  return(1);
}

main ()
{
  int   tid=-1, nth;
  double A, B, C, D;;

  gettimeofday(&t0, 0);
  #pragma omp parallel private(tid, nth)
  {
    #ifdef _OPENMP
    tid = omp_get_thread_num();
    nth = omp_get_num_threads();
    if (tid==0) printf("\n  Ejecucion en paralelo con %d hilos\n", nth);
    #endif

    #pragma omp sections
    {
      #pragma omp section
      {
        A = func(NA);
      }
      #pragma omp section
      {
        B = func(NB);
      }
      #pragma omp section
      {
        C = func(NC);
      }
    }
  }
  D = A + B + C;

  gettimeofday(&t1, 0);
  printf ("\n  Resultados A = %.2f   B = %.2f   C = %.2f --->  D = %.2f", A,B,C,D);

  tej = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec - t0.tv_usec) / 1e6;
  printf("\n\n  T. de ejec. = %1.3f ms\n\n", tej*1000);
}
```

```
/******************************************************************************
   orf.c     OPENMP
   calculo de un producto escalar
   reparto de tares tipo orphan
******************************************************************************/
//// Ejecutar la versión serie (orf_ser) antes para ver lo que tiene que dar

#include <omp.h>
#include <stdio.h>

#define N 100
float A[N], B[N], pe;


void AporB()
{
  int j;

  #pragma omp for reduction(+:pe)
  for (j=0; j<N; j++) pe += A[j] * B[j];
}


main ()
{
  int i;

  for (i=0; i<N; i++)
  {
    A[i] = i;
    B[i] = N-i;
  }
  pe = 0.0;

  #pragma omp parallel
  {
    AporB();
  }

  printf("\n\n     >> PE = %10.0f\n\n", pe);
}
```

```
/******************************************************************************
   nested.c
   Ejemplo de regiones paralelas anidadas
******************************************************************************/

#ifdef _OPENMP
  #include <omp.h>
  #define TRUE  1
  #define FALSE 0
#else
  #define omp_get_thread_num() 0
  #define omp_get_num_threads() 1
  #define omp_get_nested() 0
  #define omp_get_level() 0
#endif

int main()
{

int tid = -1;

#ifdef _OPENMP
   (void) omp_set_num_threads(4);

   (void) omp_set_nested(TRUE);
   if (! omp_get_nested()) {printf("Cuidado! No se ha activado el paralelismo
anidado \n");}
#endif

   printf("Paralelismo anidado esta %s\n",
           omp_get_nested() ? "activado" : "no activado");

#pragma omp parallel private (tid)
   {
     tid = omp_get_thread_num();
     printf("TID(%d) ejecuta la region externa (nivel: %d)\n",tid,omp_get_level());

     #pragma omp parallel num_threads(3)
     {
        printf(" >>>TID(%d.%d): El thread %d ejecuta la region interna (nivel:
%d)\n",
               tid,omp_get_thread_num(),omp_get_thread_num(),omp_get_level());
     }
   }

  return(0);
}
```

Ejercicio: `histo.c`

Paralelizar un programa que efectúa unas cuantas operaciones sobre una matriz (una "imagen"):

1. Calcula el histograma de la imagen (datos iniciales: de 0 a `NG`).
2. Calcula el valor mínimo del histograma ("nivel de gris que menos veces aparece").
3. A partir de ahí, calcula:
   - `B`: vector que contiene la suma de los elementos de cada fila hasta (sin incluir) encontrar el valor mínimo
   - `C`: vector que contiene la posición del mínimo en cada fila
   - `SPM`: suma de las posiciones del mínimo

En una primera versión usar una imagen de tamaño 10 × 10 y una valor de NG de 10.

```
/******************************************************************************
     histo_ser.c
******************************************************************************/

#include <omp.h>
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>

#define N  10    /* luego 6000  */
#define NG 10    /* luego 256   */

main ()
{
  struct timeval        t0, t1;
  double                tej;

  int  IMA[N][N], histo[NG], B[N], C[N];
  int  i, j, tid, hmin, imin, spm = 0, x;

// Inicializacion de variables (aleatorio)
  for(i=0; i<N; i++)
    for(j=0; j<N; j++) IMA[i][j] = rand() % NG;

// se imprimen 10 elementos de 10 filas de IMA
  printf("\n  Matriz IMA ");
  for(i=0;i<10;i++)
  {
    printf("\n");
    for (j=0; j<10; j++) printf(" %3d", IMA[i][j]);
    printf("\n");
  }

  for(i=0; i<NG; i++) histo[i] = 0;

// toma de tiempos
  gettimeofday(&t0, 0);


// 1. Calculo del histograma de IMA
  for (i=0; i<N; i++)
    for (j=0; j<N; j++)
        histo[IMA[i][j]] ++;
```

```
// 2. Busqueda del minimo del histograma
  hmin = N*N+1;
  for (i=0; i<NG; i++)
   if (hmin > histo[i])
    { hmin = histo[i]; imin = i; }

// Calculo de B, C y SPM
  for (i=0; i<N; i++)
  {
    j = 0;
    x = 0;
    while ((IMA[i][j] != imin) && (j<N))
    {
      x = x + IMA[i][j] ;
      j++;
    }
    B[i] = x;
    C[i] = j;
    spm = spm + j;
  }


// toma de tiempos

  gettimeofday(&t1, 0);


// Imprimir resultados

  printf("\n Histograma \n");
  for(i=0; i<10; i++) printf("%5d",i);
  printf("\n");
  for(i=0; i<10; i++) printf("%5d",histo[i]);

  printf("\n hmin = %d  imin = %d\n", hmin, imin);

  printf("\n   Vector B \n");
  for(i=0;i<10;i++) printf(" %3d", B[i]);
  printf("\n   Vector C \n");
  for(i=0;i<10;i++) printf(" %3d", C[i]);

  printf("\n  SPM = %d\n\n", spm);

//  tej = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec - t0.tv_usec) / 1e6;
//  printf("\n T. ejec. (serie) = %1.3f ms \n\n", tej*1000);
}
```

```
/******************************************************************************
   sincro1.c
   Ejemplo para ver como sincronizar un bucle
        B(i) = B(i) + 3
        C(i) = B(i-1) * 5
   sincronizacion mediante un contador.  MAL: se bloquea.
******************************************************************************/

#include <stdio.h>

#ifdef _OPENMP
  #include <omp.h>
  #define TRUE  1
  #define FALSE 0
#else
  #define omp_get_thread_num()  0
  #define omp_get_num_threads() 1
#endif

#define N 17

main ()
{
  int   tid, i;
  int   C[N], suma;
  int  B[N];
  int KB;

// inicializacion de datos
  for (i=0; i<N; i++)
  {
    B[i] = i;
    C[i] = 0;
  }
  KB = 0;                       // contador de sincronizacion

  #pragma omp parallel shared(B,C, KB) private(tid)
  {
    tid = omp_get_thread_num();
    #pragma omp for schedule(static,1)
    for (i=1; i<N; i++)
    {
      B[i] = B[i] + 3;
      printf("\n >> Thread %d esperando a que KB = %d", tid, i-1);
      while (KB < (i-1)) ;      // espera para sincronizacion wait(KB,i-1)
      KB = i;                   // post(KB,i)
      C[i] = B[i-1] * 5;
    }
  }

  suma = 0;
  for (i=0; i<N; i++) suma += C[i];
  printf("\n\n\n C(%d) es %d   KB es %d \n\n", N-1, C[N-1], KB);
  printf("\n  Valores de B  ");
  for (i=0; i<N; i++) printf(" %d ", B[i]);
  printf("\n\n");
  printf("\n  Valores de C  ");
  for (i=1; i<N; i++) printf(" %d ", C[i]);
  printf("\n\n");
  printf ("Suma de C es: %d\n\n", suma);
}
```

```
/*******************************************************
  fibo_ser.c (usleep)
  Ejemplo de uso de la directiva task en un programa
  recursivo para calcular el numero de fibonacci
  f(n) = f(n-1) + f(n-2) con f(0)=1 y f(1)=1
        version serie con usleep
*******************************************************/

#include <stdio.h>
#include <sys/time.h>
#include <unistd.h>

long fibonacci(int n)
{
 long fn1, fn2, fn;

 usleep(2);

 if ( n == 0 || n == 1 ) return(1);

 fn1 = fibonacci(n-1);
 fn2 = fibonacci(n-2);
 fn = fn1 + fn2;

 return(fn);
 }

main  ()
{
  struct timeval t0, t1;
  double         tej;
  int     n;
  long  resul;

  printf("\n Numero a calcular?  ");
  scanf("%d", &n);

  gettimeofday(&t0, 0);
  resul = fibonacci(n);
  gettimeofday(&t1, 0);

  printf ("\nEl numero Fibonacci de %d es %d", n, resul);

  tej = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec - t0.tv_usec) / 1e6;
  printf("\n T. de ejec. = %1.3f ms \n\n", tej*1000);
}
```

```
/*********************************************************
  fibo.c
  Ejemplo de uso de la directiva task en un programa
  recursivo para calcular el numero de fibonacci
        (version paralela con sleep)
*********************************************************/
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>

#ifdef _OPENMP
  #include <omp.h>
  #define TRUE  1
  #define FALSE 0
#else
  #define omp_get_thread_num()  0
  #define omp_get_num_threads() 1
#endif

long fibonacci(int n)
{
  long fn1, fn2, fn;

  usleep(1);

  if ( n == 0 || n == 1 ) return(1);

//  if ( n < 30 ) return(fibonacci(n-1) + fibonacci(n-2));

 #pragma omp task shared(fn1)
  fn1 = fibonacci(n-1);

 #pragma omp task shared(fn2)
  fn2 = fibonacci(n-2);

 #pragma omp taskwait
  fn = fn1 + fn2;

  return(fn);
 }

main  ()
{
  struct timeval t0, t1;
  double         tej;
  int nthr=0;
  int     n;
  long  resul;

  printf("\n Numero a calcular?  ");
  scanf("%d", &n);

  gettimeofday(&t0, 0);

 #pragma omp parallel shared (resul)
  {
   #pragma omp single
    {
     resul = fibonacci(n);
    }
  }

  gettimeofday(&t1, 0);
  printf ("\nEl numero Fibonacci de %5d es %d", n, resul);

  tej = (t1.tv_sec - t0.tv_sec) + (t1.tv_usec - t0.tv_usec) / 1e6;
  printf("\n T. de ejec. = %1.3f ms \n\n", tej*1000);
}
```

```
/******************************************************************************
    scri.c
    uso de una seccion critica para planificacion guided de un bucle
    se reparte un cuarto de la parte proporcional
    se hacen N iteraciones en las que una de ellas, N/4, es mucho mas larga
      POR COMPLETAR
******************************************************************************/
//// Esquema dado para teminar

#include <omp.h>
#include <math.h>
#include <stdlib.h>
#include <stdio.h>
#include <unistd.h>

#define N 1000

int calculo(int T)
{
  usleep(10*T);
  return(rand()%100);
}

main ()
{
  int A[N];
  int j, ini, fin, tid, nthr;

  int  larga;  // Variable para almacenar el tid del thread que ejecuta la larga

  int  ITER[10]; // En ITER[tid] el numero de iter. ejecutadas por el thread tid

  for (j=0; j<N; j++) A[j] = rand() % 100;
  A[N/4] = 10000;
  for (j=0; j<10; j++) ITER[j] = 0;

  #pragma omp parallel
  {
    tid = omp_get_thread_num();
    nthr = omp_get_num_threads();


    /* OPERACION DE PLANIFICACION */
    /* Dejar en ini y fin las iteraciones comienzo y fin */

    while (ini<N)
    {

     /* EJECUTAR TRABAJO PLANIFICADO */
     /* Por cada iteracion hacer: if (A[x] > 90) A[x] = calculo(A[x]); */

     /* OPERACION DE PLANIFICACION */
     /* Dejar en ini y fin las iteraciones comienzo y fin */

    }
  }

  for (j=0; j<nthr; j++) printf("\n IT(%d) = %d",j,  ITER[j]);
  printf("\n\n larga --> %d\n\n", larga);
}
```