Programación Avanzada (TC2025)

Tema 6. Programación multinúcleo

Instituto Tecnológico y de Estudios Superiores de Monterrey. Campus Santa Fe
Departamento de Tecnologías de Información y Electrónica
Dr. Vicente Cubells (vcubells@itesm.mx)

Temario

- Introducción a OpenMP
- Modelo de programación OpenMP
- Directivas
- Funciones de la RTL
- Variables de entorno

¿Qué es OpenMP?

- API para programar aplicaciones paralelas multihilos con memoria compartida
- Compuesta por 3 elementos
  - Directivas del compilador (44)
  - Funciones RTL (35)
  - Variables de entorno (13)
- Portable: C/C++ y Fortran
- Estandarizada
- Bibliografía
  - openmp.org

Qué no es

- Mecanismo para el desarrollo de aplicaciones con memoria distribuida
- La implementación de diferentes fabricantes puede variar
- No garantiza el uso más eficiente de la memoria compartida
- Requerimiento para comprobar dependencias de datos, conflictos, condiciones de competencia y bloqueos
- Mecanismo para proveer paralelización automática
- Diseñado para garantizar entradas y salidas síncronas

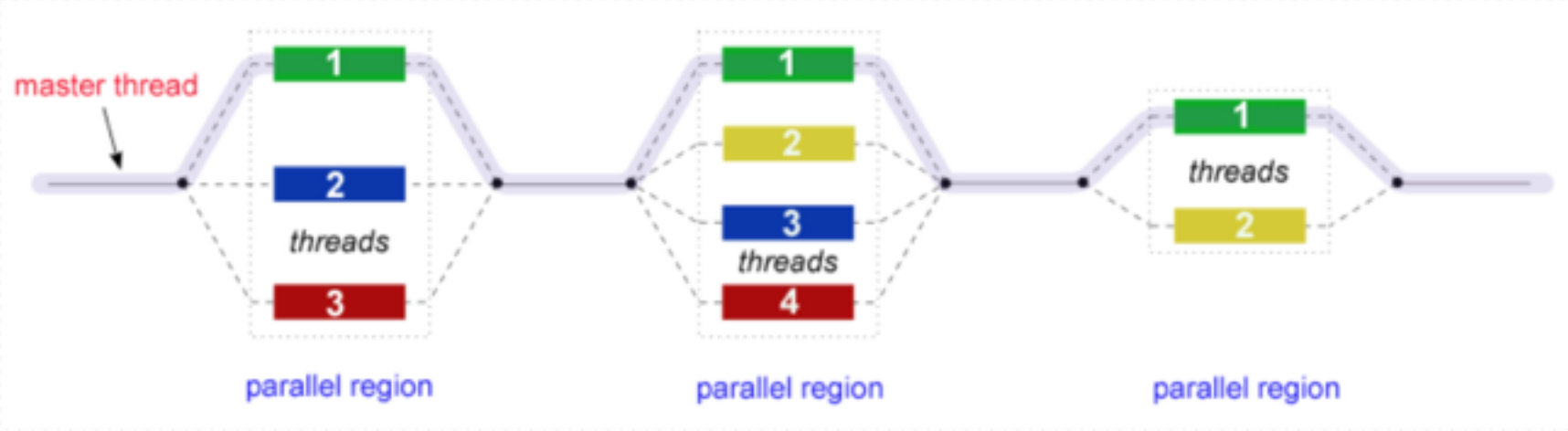
Versiones

Month/Year	Version
Oct 1997	Fortran 1.0
Oct 1998	C/C++ 1.0
Nov 1999	Fortran 1.1
Nov 2000	Fortran 2.0
Mar 2002	C/C++ 2.0
May 2005	OpenMP 2.5
May 2008	OpenMP 3.0
Jul 2011	OpenMP 3.1
Jul 2013	OpenMP 4.0

https://computing.llnl.gov/tutorials/openMP/

Modelo de programación…

- Memoria compartida, paralelismo basado en hilos
- Paralelismo explícito
- Modelo fork-join



https://computing.llnl.gov/tutorials/openMP/

Modelo de programación

- Basado en directivas del compilador
  - **omp parallel**
- Soporte de paralelismo anidado
  - Regiones paralelas dentro de regiones paralelas
  - Depende de la implementación
- Hilos dinámicos
  - Uso eficiente de los recursos siempre que sea posible
  - Depende de la implementación
- No especifica nada sobre E/S paralelas
  - R/W simultáneas en un mismo archivo
- Caché en cada hilo
  - No es obligatorio que mantengan una consistencia exacta con la memoria real
  - Si se requiere, es responsabilidad del programador

Un ejemplo

```
#include <omp.h>

main ()  {

  int var1, var2, var3;

  //Código secuencial

 //Comienza región paralela, se crea un conjunto de hilos
 // y se define el ámbito de las variables

#pragma omp parallel private(var1, var2) shared(var3)
  {

  //Sección paralela ejecutada por todos los hilos

  } // Los hilos se unen al hilo principal

//Código secuencial

}
```

Directivas…La base

- **`#pragma omp`**
  - Requerido por todas las directivas OpenMP

- Ejemplo:

  - **`#pragma omp parallel default(shared) private(beta,pi)`**

Directivas…Construir una región paralela…

```
#pragma omp parallel [clause ...]  newline
                     if (scalar_expression)
                     private (list)
                     shared (list)
                     default (shared | none)
                     firstprivate (list)
                     reduction (operator: list)
                     copyin (list)
                     num_threads (integer-
expression)

   structured_block
```

- ¿Cuántos hilos?
  - Cláusula **IF**
  - Cláusula **NUM_THREADS**
  - Función: **omp_set_num_threads()**
  - Variable de entorno: **OMP_NUM_THREADS**
  - Definido por la implementación: **Número de CPUs en un nodo**
  - Se enumeran del **0 .. N-1**

Directivas…Construir una región paralela…

- Para determinar si los hilos dinámicos están habilitados
  - **`omp_get_dynamic()`**

- Se habilitan con:
  - **`omp_set_dynamic()`**
  - Variable entorno **OMP_DYNAMIC** = TRUE

Directivas…Construir una región paralela…

- Comprobar regiones paralelas anidadas
  - **`omp_get_nested()`**

- Se habilitan con:
  - **omp_set_nested()**
  - Variable de entorno **OMP_NESTED** = TRUE

Directivas…Construir una región paralela

- Cláusula IF
  - Si está presente debe evaluarse como True (en C != 0) para que se cree el conjunto de hilos
  - Sólo se permite una

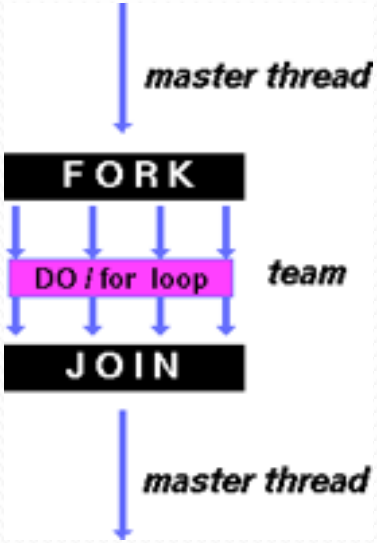
- Sólo una cláusula NUM_TREAHDS es permitida

Un ejemplo

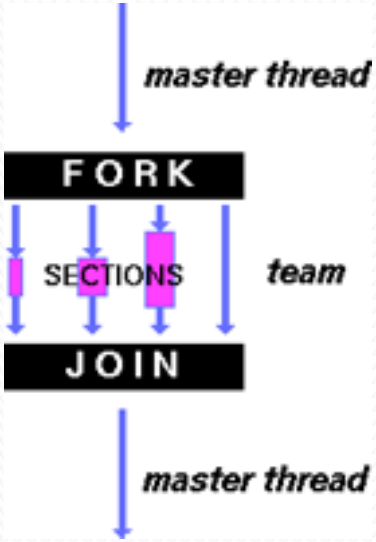
```
#include <omp.h>

main ()
{
  int nthreads, tid;

/* Cada hilo tienen una variable tid privada */
#pragma omp parallel private(tid)
  {

  /* Obtener el thread id */
  tid = omp_get_thread_num();
  printf("Soy el hilo = %d\n", tid);

  /* Solo el master lo ejecuta*/
  if (tid == 0)
    {
    nthreads = omp_get_num_threads();
    printf("Número de hilos = %d\n", nthreads);
    }
  }
}
```

Directivas…Construir una región de trabajo compartido…

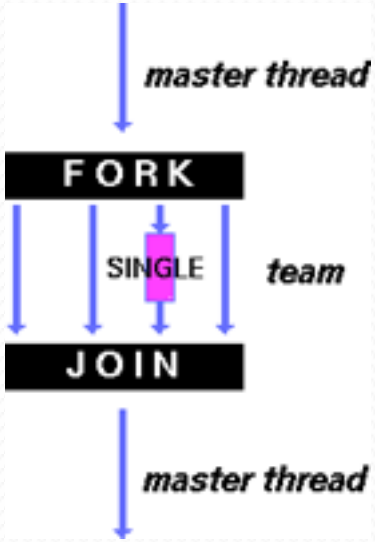
- Se divide el trabajo entre los hilos del conjunto
- No crea nuevos hilos
- No implica una barrera para entrar en la región compartida pero si a la salida



For (paralelismo de datos)



SECTIONS



SINGLE

Directivas…Construir una región de trabajo compartido…FOR

```
#pragma omp for [clause ...]  newline
                schedule (type [,chunk])
                ordered
                private (list)
                firstprivate (list)
                lastprivate (list)
                shared (list)
                reduction (operator: list)
                collapse (n)
                nowait

   for_loop
```

- **SCHEDULE**
  - STATIC
  - DYNAMIC
  - GUIDED: iteraciones/hilos
  - RUNTIME: OMP_SCHEDULE
  - AUTO
- **nowait**
- **ORDERED**
- **COLLAPSE**

Un ejemplo

```
#include <omp.h>
#define CHUNKSIZE 100
#define N 1000

main ()
{
  int i, chunk;
  float a[N], b[N], c[N];

  /* Inicializar arreglos */
  for (i=0; i < N; i++)
    a[i] = b[i] = i * 1.0;

chunk = CHUNKSIZE;

#pragma omp parallel shared(a,b,c,chunk) private(i)
  {

  #pragma omp for schedule(dynamic,chunk) nowait
  for (i=0; i < N; i++)
    c[i] = a[i] + b[i];

  }
}
```

Directivas…Construir una región de trabajo compartido…SECTIONS

```
#pragma omp sections [clause ...]  newline
                     private (list)
                     firstprivate (list)
                     lastprivate (list)
                     reduction (operator: list)
                     nowait
  {
  #pragma omp section    newline
     structured_block

 #pragma omp section    newline
     structured_block

  }
```

- ¿Más hilos que secciones?
- ¿Menos hilos que secciones?
- ¿Qué hilo ejecuta cuál sección?

Un ejemplo

```
#include <omp.h>
#define N 1000

main ()
{
  int i;
  float a[N], b[N], c[N], d[N];

/* Inicialización */
for (i=0; i < N; i++) {
  a[i] = i * 1.5;
  b[i] = i + 22.35;
}
```

```
#pragma omp parallel shared(a,b,c,d) private(i)
  {

  #pragma omp sections nowait
    {

    #pragma omp section
    for (i=0; i < N; i++)
      c[i] = a[i] + b[i];

    #pragma omp section
    for (i=0; i < N; i++)
      d[i] = a[i] * b[i];

    }  /* fin secciones */

  }  /* fin paralela */

}
```

Directivas…Construir una región de trabajo compartido…SINGLE

```
#pragma omp single [clause ...]  newline
                   private (list)
                   firstprivate (list)
                   nowait

     structured_block
```

- Solo un hilo lo ejecuta
- Para funciones no seguras
  - E/S
- Los demás hilos esperan
  - Si no **`nowait`**

Directivas…Sincronización…

- MASTER

```
#pragma omp master  newline

   structured_block
```

- CRITICAL
  - `name` es un identificador global

```
#pragma omp critical [ name ]  newline

   structured_block
```

Un ejemplo

```
#include <omp.h>

main()
{

  int x;
  x = 0;

#pragma omp parallel shared(x)
  {

  #pragma omp critical
  x = x + 1;

  }
}
```

Directivas…Sincronización

- BARRIER

```
#pragma omp barrier  newline
```

- ATOMIC

```
#pragma omp atomic  newline
   statement_expression
```

- FLUSH

```
#pragma omp flush (list)  newline
```

Directivas…THREADPRIVATE

- Valores de variables globales persisten entre diferentes regiones paralelas

```
#include <omp.h>

int  a, b, i, tid;
float x;

#pragma omp threadprivate(a, x)

main ()  {

/* Deshabilitar hilos dinámicos */
  omp_set_dynamic(0);

  printf("Región paralela 1:\n");
#pragma omp parallel private(b,tid)
  {
  tid = omp_get_thread_num();
  a = tid;
  b = tid;
  x = 1.1 * tid +1.0;
  printf("Thread %d:   a,b,x= %d %d %f\n",tid,a,b,x);
  }  /* fin paralela 1 */

  printf("************************************\n");
  printf("   Trabajo secuencial del master    \n");
  printf("************************************\n");

  printf("Región paralela 2:\n");
#pragma omp parallel private(tid)
  {
  tid = omp_get_thread_num();
  printf("Thread %d:   a,b,x= %d %d %f\n",tid,a,b,x);
  }  /* fin paralela 2 */

}
```

Directivas…Definiendo el ámbito de las variables

- PRIVATE: Privadas a cada hilo (no persistentes)
  - `private (list)`
- FIRSTPRIVATE: Se inicializan con el valor de la variable original
  - `firstprivate (list)`
- LASTPRIVATE: Se copia el último valor a la variable original
  - `lastprivate (list)`
- SHARED: Compartidas entre todos los hilos
  - `shared (list)`
- DEFAULT: ámbito predeterminado para todas las variables a las cuales no se les especifique explícitamente
  - `default (shared | none)`
- REDUCTION: Realiza una operación de reducción con las copias privadas de todos los hilos
  - `reduction (operator: list)`
- COPYIN: Asigna el mismo valor a las variables THREADPRIVATE de todos los hilos (el de la variable original)
  - `copyin (list)`

Directivas…Haciendo reducciones

```
#include <omp.h>

main ()  {
  int   i, n, chunk;
  float a[100], b[100], result;

  n = 100;
  chunk = 10;
  result = 0.0;
  for (i=0; i < n; i++)
  {
    a[i] = i * 1.0;
    b[i] = i * 2.0;
  }

#pragma omp parallel for      \
  default(shared) private(i)  \
  schedule(static,chunk)      \
  reduction(+:result)

  for (i=0; i < n; i++)
    result = result + (a[i] * b[i]);
/* fin paralela */
  printf("Suma total = %f\n",result);
}
```

Directivas…Reducciones

- ***x = x op expr***
- ***x = expr op x*** *(excepto -)*
- ***x binop = expr***
- ***x++***
- ***++x***
- ***x--***
- ***--x***

- ***x*** variable escalar en a lista
- ***expr*** expresión escalar que no referencia ***x***
- ***op*** no es sobrecargado, y solo puede ser:

  +, *, -, /, &, ^, |, &&, ||
- ***binop*** no es sobrecargado, y solo puede ser:

  +, *, -, /, &, ^, |

Directivas…Resumen de cláusulas

Clause	Directive					
	PARALLEL	DO/for	SECTIONS	SINGLE	PARALLEL DO/for	PARALLEL SECTIONS
IF	●				●	●
PRIVATE	●	●	●	●	●	●
SHARED	●	●			●	●
DEFAULT	●				●	●
FIRSTPRIVATE	●	●	●	●	●	●
LASTPRIVATE		●	●		●	●
REDUCTION	●	●	●		●	●
COPYIN	●				●	●
COPYPRIVATE				●		
SCHEDULE		●			●	
ORDERED		●			●	
NOWAIT		●	●	●		

Funciones de la RTL…

- **omp.h** en C/C++
- Obtener el número de hilos/procesadores
- Establecer el número de hilos a usar
- Temporización con relojes
- Paralelismo anidado
- Ajuste dinámico de los hilos
- Semáforos
- …

Funciones de la RTL…

- `#include <omp.h>`
- `void omp_set_num_threads(int num_threads)`
- `int omp_get_num_threads(void)`
- `int omp_get_max_threads(void)`
- `int omp_get_thread_num(void)`
- `int omp_get_thread_limit (void)`
- `int omp_get_num_procs(void)`
- `int omp_in_parallel(void)`
- `void omp_set_dynamic(int dynamic_threads)`
- `int omp_get_dynamic(void)`
- `void omp_set_nested(int nested)`
- `int omp_get_nested (void)`

Funciones de la RTL…

- `#include <omp.h>`
- `void omp_init_lock(omp_lock_t *lock)`
- `void omp_init_nest_lock(omp_nest_lock_t *lock)`
- `void omp_destroy_lock(omp_lock_t *lock)`
- `void omp_destroy_nest__lock(omp_nest_lock_t *lock)`
- `void omp_set_lock(omp_lock_t *lock)`
- `void omp_set_nest__lock(omp_nest_lock_t *lock)`
- `void omp_unset_lock(omp_lock_t *lock)`
- `void omp_unset_nest__lock(omp_nest_lock_t *lock)`
- `int omp_test_lock(omp_lock_t *lock)`
- `int omp_test_nest__lock(omp_nest_lock_t *lock)`

Funciones de la RTL

- `#include <omp.h>`
- `double omp_get_wtime(void)`
- `double omp_get_wtick(void)`

- `omp_set_schedule`
  `omp_get_schedule`
  `omp_set_max_active_levels`
  `omp_get_max_active_levels`
  `omp_get_level`
  `omp_get_ancestor_thread_num`
  `omp_get_team_size`
  `omp_get_active_level`
  `omp_in_final`

Variables de entorno

- **OMP_SCHEDULE - "guided, 4" | "dynamic"**
- **OMP_NUM_THREADS - 8**
- **OMP_DYNAMIC – TRUE | FALSE**
- **OMP_PROC_BIND – TRUE | FALSE**
- **OMP_NESTED – TRUE | FALSE**
- **OMP_STACKSIZE - "3000 k " | 10M | " 1G"**
- **OMP_WAIT_POLICY – ACTIVE | PASSIVE**
- **OMP_MAX_ACTIVE_LEVELS – 2**
- **OMP_THREAD_LIMIT - 8**

Actividad práctica

- Desarrollar algunos programas que utilicen OpenMP