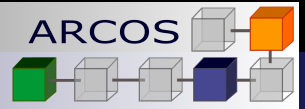
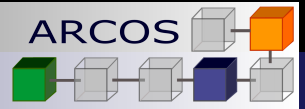


Programación paralela con OpenMP

Arquitectura de Computadores

J. Daniel García Sánchez (coordinador)
David Expósito Singh
Javier García Blas
Óscar Pérez Alonso
J. Manuel Pérez Lobato

Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid



¿Qué es OpenMP?

- Es un **API** (y un conjunto de directivas) para expresar aplicaciones paralelas en sistemas de memoria compartida.
- **Componentes**:
  - Directivas de compilador.
  - Funciones de biblioteca.
  - Variables de entorno.
- Simplifica la forma de escribir programas paralelos.
  - *Mappings* para FORTRAN, C y C++.

Construcciones

- Directivas:

  #pragma omp directiva [clausulas]

  - **Ejemplo**: Fijar el número de hilos.

    #pragma omp parallel num_threads(4)

- Funciones de biblioteca

  #include <omp.h> *// Incluir para llamar funciones OpenMP*

  - **ejemplo**: Obtener el número de hilos usado.

    int n = omp_get_num_threads();

Ejercicio 1: Secuencial

exseq.cpp

```
#include <iostream>

int  main() {
   using namespace std;

   int  id  = 0;
   cout << "Hola(" << id  << ")  ";
   cout << "Mundo(" << id << ")";
   return 0;
}
```

- Imprime en salida estándar.

Ejercicio 1: Paralelo

exseq.cpp

```
#include <iostream>
#include <omp.h>

int main() {
  using namespace std;

  #pragma omp parallel
  {
    int id = omp_get_thread_num();
    cout << "Hola(" << id << ") ";
    cout << "Mundo(" << id << ")";
  }

  return 0;
}
```

- Flags de compilación:
  - **gcc**: **-fopenmp**
  - **Intel Linux**: **-openmp**
  - **Intel Windows**: **/Qopenmp**
  - **Microsft Visual Studio**: **/openmp**

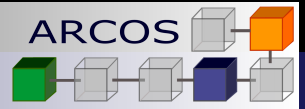
Ejercicio 1

- **Objetivo**: Verificar que el entorno funciona.

- **Actividades**:
  1. Compilar la versión secuencial y ejecutar.
  2. Compilar la versión paralela y ejecutar.
  3. Introduzca la función **omp_get_num_threads()** para imprimir el número de hilos:
     a) Antes del **pragma**.
     b) Justo después del **pragma**.
     c) Dentro del bloque.
     d) Antes de salir del programa, pero fuera del bloque.

Observaciones

- Modelo de memoria compartida multi-hilo.
  - Comunicación mediante variables compartidas.
- Compartición accidental $\rightarrow$ **condiciones de carrera**.
  - Resultado dependiente de la planificación de los hilos.
- Evitar condiciones de carrera.
  - Sincronización para evitar conflictos.
    - Coste de sincronización.
  - Modificación en patrón de accesos.
    - Minimizar sincronizaciones necesarias.



Paralelismo *fork-join*

- Aplicación secuencial con secciones paralelas:
  - **Hilo maestro**: Iniciado con el programa principal.
  - En una sección paralela se arranca un conjunto de hilos.
  - Se puede **anidar** el paralelismo.

- Una región paralela es un bloque marcado con la directiva **parallel**.

```
#pragama omp parallel
```

Selección del número de hilos

- Invocando a una función de biblioteca.

Ejemplo

```
// ...
omp_set_num_threads(4);
#pragma omp parallel
{
   // Región paralela
}
```

- Directiva OpenMP.

Ejemplo

```
// ...
#pragma omp parallel num_threads(4)
{
   // Región paralela
}
```

Ejercicio 2: Cálculo de π

- Cálculo de π.

$$\pi = \int_0^1 \frac{4}{1+x^2} dx$$

- Aproximación:

$$\pi \approx \sum_{i=0}^{N} F(x_i)\Delta x$$

  - Suma del área de N rectángulos.
  - **Base**: Δx.
  - **Altura**: $F(x_i)$.

Ejercicio 2: Versión secuencial

Cálculo de π

```
#include <iostream>
#include <iomanip>
#include <chrono>

int main() {
  using namespace std;
  using namespace std::chrono;

  constexpr long nsteps = 10000000;
  double step = 1.0 / double(nsteps);

  using clk = high_resolution_clock;
  auto t1 = clk::now();

  double sum = 0.0;
  for (int i=0;i<nsteps; ++i) {
    double x = (i+0.5) * step;
    sum += 4.0 / (1.0 + x * x);
  }
  double pi = step * sum;

  auto t2 = clk::now();
  auto diff = duration_cast<microseconds>(t2-t1);
```

Medición del tiempo en C++11

- Ficheros de **include**:

```
#include <chrono>
```

- Tipo para el reloj:

```
using clk = chrono::high_resolution_clock;
```

- Obtener un punto del tiempo:

```
auto t1 = clk::now();
```

- Diferencias de tiempo (puede especificar unidad).

```
auto diff = duration_cast<microseconds>(t2-t1);
```

- Obtención del valor de la diferencia.

```
cout << diff.count();
```

Ejemplo de medida de tiempo

Ejemplo

```
#include <chrono>

void f() {
  using namespace std;
  using namespace std::chrono;

  using clk = chrono::high_resolution_clock;

  auto t1 = clk::now();

  g();

  auto t2 = clk::now();
  auto diff = duration_cast<microseconds>(t2-t1);

  cout << "Time= " << diff.count << "microseconds" << endl;
}
```

Medición de tiempo en OpenMP

- Punto del tiempo.

```
double t1 = omp_get_wtime();
```

- Diferencia de tiempo.

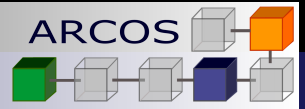
```
double t1 = omp_get_wtime();
double t2 = omp_get_wtime();
double diff = t2-t1;
```

- Diferencia de tiempo entre 2 ticks sucesivos:

```
double tick = omp_get_wtick();
```

Ejercicio 2

- Crear una versión paralela del programa de cálculo de π usando una clausula **parallel**.
- **Observaciones**:
  - Incluya mediciones de tiempo.
  - Imprima el número de hilos que se están usando.
  - Tenga cuidado con las variables compartidas.
  - **Idea**: Use un array y acumule una suma parcial en cada hilo en la región paralela.



Mecanismos de sincronización

- **Sincronización**: Mecanismo usado para establecer restricciones sobre el orden de acceso a datos compartidos.
  - **Objetivo**: Evitar carreras de datos.

- **Alternativas**:
  - **Alto nivel**: **critical**, **atomic**, **barrier**, **ordered**.
  - **Bajo nivel**: **flush**, cerrojos.

critical

- Garantiza **exclusión mutua**.
- Solamente un hilo a la vez puede entrar en la región crítica.

Ejemplo

```
#pragma omp parallel
{
  for (int i=0;i<max;++i) {
    x = f(i);
    #pragma omp critical
    g(x);
}
```

- Las llamadas a **f()** se realiza en paralelo.
- Solamente un hilo puede entrar a la vez en **g()**.

atomic

- Garantiza la **actualización atómica** de una posición de memoria.
- Evita carrera de datos en actualización a variable

Ejemplo

```
#pragma omp parallel
{
  for (int i=0;i<max;++i) {
    x = f(i);
    #pragma omp atomic
    s += g(x)
}
```

- Las llamadas a **f()** se realiza en paralelo.
- Las actualizaciones a s son *thread-safe*.

Ejercicio 3

- Modifique su programa del ejercicio 2.
- Evalúe:
  a) Sección crítica.
  b) Acceso atómico.

Parallel for

- **División de bucle**: Realiza una partición de las iteraciones de un bucle entre los hilos disponible.

Sintaxis

```
#pragma omp parallel
{
  #pragma omp for
  for (i=0; i<n; ++i) {
    f(i);
  }
}
```

- **omp for** → División de bucle for.
- Se generar una copia privada de **i** para cada hilo.
  - Se puede hacer también con **private(i)**

Ejemplo

Código secuencial

```
for (i=0;i<max;++i) { u[i] = v[i] + w[i]; }
```

Región paralela

```
#pragma omp parallel
{
  int id = omp_get_thread_num();
  int nthreads = omp_get_num_threads();
  int istart = id * max / nthreads;
  int iend = (id==nthreads-1) ?
    ((id + 1) * max / nthreads):max;
  for (int i=istart ; i<iend;++i)
    { u[i] = v[i] + w[i]; }
}
```

Región paralela + Bucle paralelo

```
#pragma omp parallel
#pragma omp for
for (i=0;i<max;++i)
  { u[i] = v[i] + w[i]; }
```

Construcción combinada

- Se puede usar una **forma abreviada** combinando las dos directivas.

Dos directivas

```
vector<double> vec(max);
#pragma omp parallel
{
  #pragma omp for
  for (i=0; i<max; ++i) {
    vec[i] = generate(i);
  }
}
```

Directiva combinada

```
vector<double> vec(max);
#pragma omp parallel for
for (i=0; i<max; ++i) {
  vec[i] = generate(i);
}
```

Reducciones

Ejemplo

```
double sum = 0.0;
vector<double> v(max);
for (int i=0; i<max; ++i) {
  suma += v[i];
}
```

- Una **reducción** es una operación de acumulación en un bucle.
- Clausula de reducción: **reduction (op:var)**

- **Efectos**:
  - Copia privada de cada variable.
  - Actualización de copia local en cada iteración.
  - Copias locales combinadas al final.

Ejemplo

```
double sum = 0.0;
vector<double> v(max);
#pragma omp parallel for reduction(+:suma)
for (int i=0; i<max; ++i) {
  suma += v[i];
}
```

Operaciones de reducción

- Operaciones que son asociativas.

$$(a \oplus b) \oplus c = a \oplus (b \oplus c)$$

- Valor inicial definido por la operación.
- **Operadores básicos**:
  - **+** (valor inicial: **0**).
  - * (valor inicial: **1**).
  - **-** (valor inicial: **0**).
- **Operadores avanzados**:
  - **&** (valor inicial: **0**).
  - | (valor inicial: **0**).
  - ^ (valor inicial: **0**).
  - **&&** (valor inicial: **1**).
  - || (valor inicial: **0**).

Ejercicio 4

- Modifique el programa de cálculo de π.
- Intente que el programa sea lo más parecido posible al programa secuencial.

Barreras

- Permite sincronizar todos los hilos en un punto.
  - Se espera hasta que todos los puntos llegan a la **barrera**.

Ejemplo

```
#pragma omp parallel
{
  int id = omp_get_thread_num();
  v[id] = f(id);
  #pragma omp barrier

  #pragma omp for
  for (int i=0;i<max;++i) {
    w[i] = g(i);
  } // Barrera implícita

  #pragma omp for nowait
  for (int i=0;i<max;++i) {
    w[i] = g(i);
  } // nowait -> No hay barrera implícita

  v[i] = h(i);
} // Barrera implícita
```

Ejecución única: master

- La clausula **master** marca un bloque que solamente se ejecuta en el hilo *maestro*.

Ejemplo

```
#pragma omp parallel
{
  f(); // En todos los hilos
  #pragma omp master
  {
    g(); // Solamente en maestro
    h(); // Solamente en maestro
  }
  i(); // En todos los hilos
}
```

Ejecución única: single

- La clausula **single** marca un bloque que solamente se ejecuta en un hilo.
  - No tiene por qué ser el hilo maestro.

Ejemplo

```
#pragma omp parallel
{
  f(); // En todos los hilos
  #pragma omp single
  {
    g(); // Solamente en algún hilo
    h(); // Solamente en algún hilo
  }
  i(); // En todos los hilos
}
```

Ordenación

- Una región **ordered** se ejecuta en orden secuencial.

Ejemplo

```
#pragma omp parallel
{
  #pragma omp for ordered reduction(+:res)
  for (int i=0;i<max;++i) {
    double tmp = f(i);
    #pragma ordered
    res += g(tmp);
  }
}
```

Cerrojos simples

- Cerrojos de la biblioteca OpenMP.
  - También hay cerrojos anidados.

Ejemplo

```
omp_lock_t l;
omo_init_lock(&l);

#pragma omp parallel
{
   int  id  = omp_get_thread_num();
   double x = f(i);
   omp_set_lock(&l);
   cout << "ID=" << id  << "  tmp= " << tmp << endl;
   omp_unset_lock(&l);
}
omp_destroy_lock(&l);
```

Otras funciones de biblioteca

- **Cerrojos anidados**:
  - **omp_init_nest_lock()**, **omp_set_nest_lock()**, **omp_unset_nest_lock()**, **omp_test_next_lock()**, **omp_destroy_nest_lock()**.
- **Consulta de procesadores**:
  - **omp_num_procs()**.
- **Número de hilos**:
  - **omp_set_num_threads()**, **omp_get_num_threads()**, **omp_get_thread_num()**, **omp_get_max_threads()**.
- **Comprobación de región paralela**:
  - **omp_in_parallel()**.
- **Selección dinámica de número de hilos**:
  - **omp_set_dynamic()**, **omp_get_dynamic()**.

Variables de entorno

- Número de hilos por defecto:
  - **OMP_NUM_THREADS**
- Modo de planificación:
  - **OMP_SCHEDULE**

Atributos de almacenamiento

- Modelo de programación en **memoria compartida**:
  - **Variables compartidas**.
  - **Variables privadas**.
- **Compartidas**:
  - Variables globales (alcance de fichero y espacio de nombres).
  - Variables **static**.
  - Objetos en memoria dinámica (**malloc()** y **new**).
- **Privadas**:
  - Variables locales de funciones invocadas desde una región paralela.
  - Variables locales definidas dentro de un bloque.

Modificación de atributos de almacenamiento

- Atributos en clausulas paralelas:
  - **shared**.
  - **private**.
  - **firstprivate**.
- **private** crea una nueva copia local por hilo.
  - El valor de las copias no se inicializa.

Ejemplo

```
void f() {
  int x = 17;
  #pragma omp parallel for private(x)
  for (int i=0;i<max;++i) {
    x += i;   // x inicialmente vale 17
  }
  cout << x << endl; // x==17
}
```

firstprivate

- Caso particular de **private**.
  - Cada copia privada se inicia con el valor de la variable en el hilo **maestro**.

Ejemplo

```
void f() {
  int x = 17;
  #pragma omp parallel for firstprivate (x)
  for (long i=0;i<maxval;++i) {
    x += i;   // x inicialmente vale 17
  }
  std::cout << x << std::endl;  // x==17
}
```

lastprivate

- Pasa el valor de una variable privada de la última iteración **secuencial** a la variable global.

Ejemplo

```
void f() {
  int x = 17;
  #pragma omp parallel for firstprivate (x) lastprivate (x)
  for (long i=0;i<maxval;++i) {
    x += i; // x inicialmente vale 17
  }
  std::cout << x << std::endl; // x valor iteración i==maxval-1
}
```

Secciones

- Define un conjunto de secciones de código.
- Cada sección se pasa a un hilo distinto.
- Por defecto hay una barrera al final del bloque **sections**

Ejemplo

```
#pragma omp parallel
{
  #pragma omp sections
  {
    #pragma omp section
    f();
    #pragma omp section
    g();
    #pragma omp section
    h();
  }
}
```

Planificación de bucles

- **schedule(static)** | **schedule(static,n)**:
  - Planifica bloques de iteraciones (de tamaño *n*) para cada hilo.
- **schedule(dynamic)** | **schedule(dynamic,n)**:
  - Cada hilo toma un bloque de *n* iteraciones de una cola hasta que se han procesado todas.
- **schedule(guided)** | **schedule(guided,n)**:
  - Cada hilo toma un bloque iteraciones hasta que se han procesado todas. Se comienza con un tamaño de bloque grande y se va reduciendo hasta llegar a un tamaño *n*.
- **schedule(runtime)** | **schedule(runtime,n)**:
  - Se usa lo indicado en **OMP_SCHEDULE** o por la biblioteca en tiempo de ejecución.

Resumen

- **OpenMP** permite anotar código secuencial para hacer uso de **paralelismo fork-join**.
  - Basado en el concepto de región paralela.
- Los mecanismos de sincronización pueden ser de **alto nivel** o **bajo nivel**.
- Los bucles paralelos combinados con las reducciones permiten preservar el código original de muchos algoritmos.
- Los **atributos de almacenamiento** permiten controlar las copias y compartición de datos con las regiones paralelas.
- OpenMP ofrece varios tipos de planificación.

Referencias

- Libros:
  - *An Introduction to Parallel Programming*. P. Pacheco. Morgan Kaufmann, 2011. (Cáp 5).
  - Multicore and GPU Programming. G. Barlas. Morgan Kaufmann. 2014. (Cáp 4).

- Páginas Web:
  - OpenMP: `http://www.openmp.org`.
  - Lawrence Livermore National Laboratory Tutorial: `https://computing.llnl.gov/tutorials/openMP/`.

Programación paralela con OpenMP

Arquitectura de Computadores

J. Daniel García Sánchez (coordinador)
David Expósito Singh
Javier García Blas
Óscar Pérez Alonso
J. Manuel Pérez Lobato

Grupo ARCOS
Departamento de Informática
Universidad Carlos III de Madrid