Programación paralela

Introducción a OpenMP

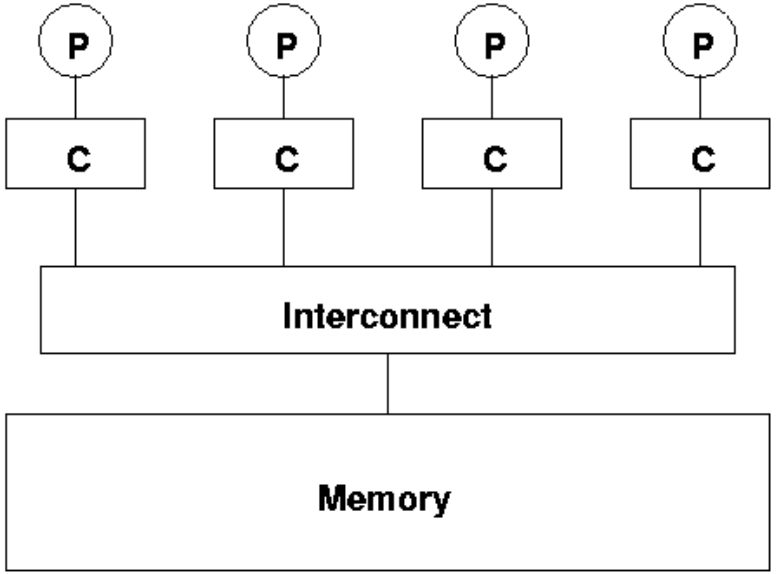
José M. Cecilia

UCAM | Universidad Católica de Murcia

¿Qué es OpenMP?

- ✓ Modelo de programación paralela
- ✓ Paralelismo de memoria compartida
- ✓ Soporta el paralelismo por datos
- ✓ Escalable
- ✓ Permite paralelización incremental
- ✓ Extensiones a lenguajes de programación existentes (Fortran, C, C++)

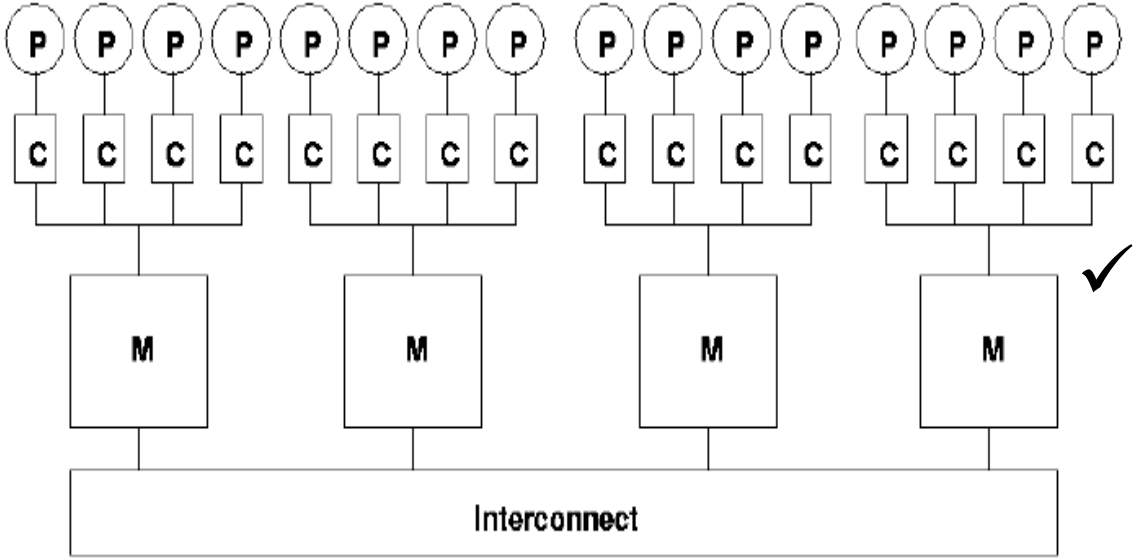
Sistemas de Memoria Compartida



Shared memory

- ✓ Programación de estos sistemas:
  - ✓ OpenMP.
  - ✓ Pthreads.

Cluster de SMPs



- ✓ Programación de estos sistemas:
  - ✓ MPI
  - ✓ OpenMP + MPI

Sintaxis de OpenMP

- ✓ La mayoria de las construcciones en OpenMP son directivas de compilación o pragmas.
- ✓ En C y C++, los pragmas tienen la forma:
  - ✓ #pragma omp construct [clause [clause]…]
- ✓ Como las construcciones son directivas, un programa en OpenMP puede ser compilado por compiladores que no soportan OpenMP.

Programa sencillo

- ✓ La mayoría de las construcciones son directivas de compilación o pragmas
- ✓ La parte central de OpenMP es la paralelización de lazos

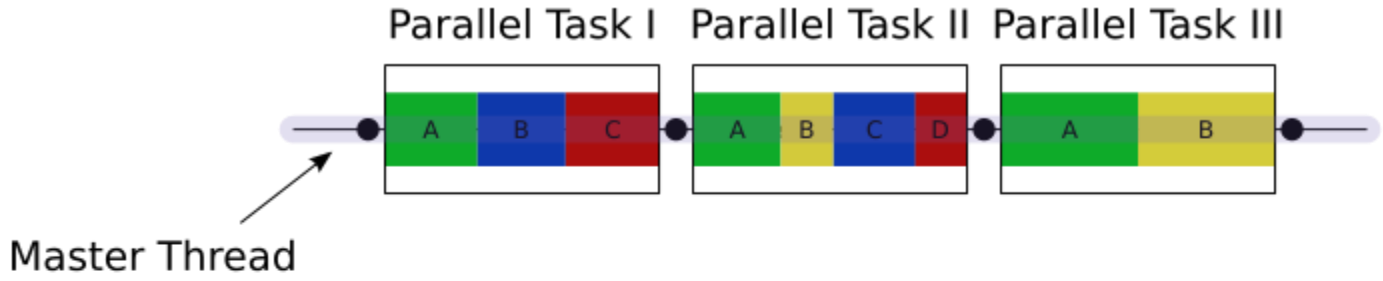
Programa Secuencial

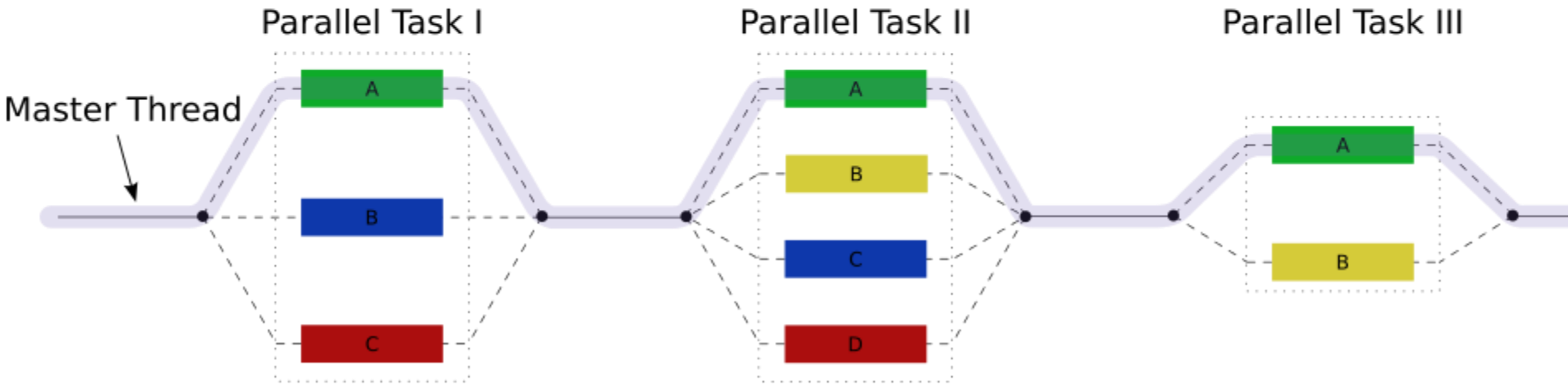
```
void main() {
   double a[1000],b[1000],c[1000];
   for (int i = 0; i< 1000; i++){
        a[i] = b[iI] + c[i];
   }
}
```

Programa Paralelo

```
void main() {
   double a[1000],b[1000],c[1000];
   #pragma omp parallel for
   for (int i = 0; i< 1000; i++){
        a[i] = b[iI] + c[i];
   }
}
```

Modelo de programación OpenMP: Fork + join





¿Cómo interactúan los threads?

- ✓ OpenMP es un modelo de memoria compartida.
  - Los threads se comunican utilizando variables compartidas.
- ✓ El uso inadecuado de variables compartidas origina condiciones de carreras.
- ✓ Para controlar las condiciones de carrera:
  - Uso de sincronización para protegerse de los conflictos de datos.
- ✓ La sincronización es costosa:
  - Modificar cómo se almacenan los datos para minimizar la necesidad de sincronización.

Alcance de los Datos

- ✓ SHARED
  - La variable es compartida por todos los procesos.
- ✓ PRIVATE
  - Cada proceso tiene una copia de la variable.

```
#pragma omp parallel for shared(a,b,c,n) private(i,temp)
for (i = 0; i < n; i++) {
    temp = a[i] / b[i];
    c[i] = temp + temp * temp;
}
```

FIRSTPRIVATE / LASTPRIVATE

- ✓ FIRSTPRIVATE
  - Las copias privadas de las variables se inicializan con los objetos originales.
- ✓ LASTPRIVATE
  - Al salir de una región privada o lazo, la variable tiene el valor que tendría en caso de una ejecución secuencial.

```
A = 2.0
#pragma omp parallel for FIRSTPRIVATE(A) LASTPRIVATE(i)
for (i = 0; i < n; i++) {
   Z[i] = A * X[i] + Y[i];
}
```

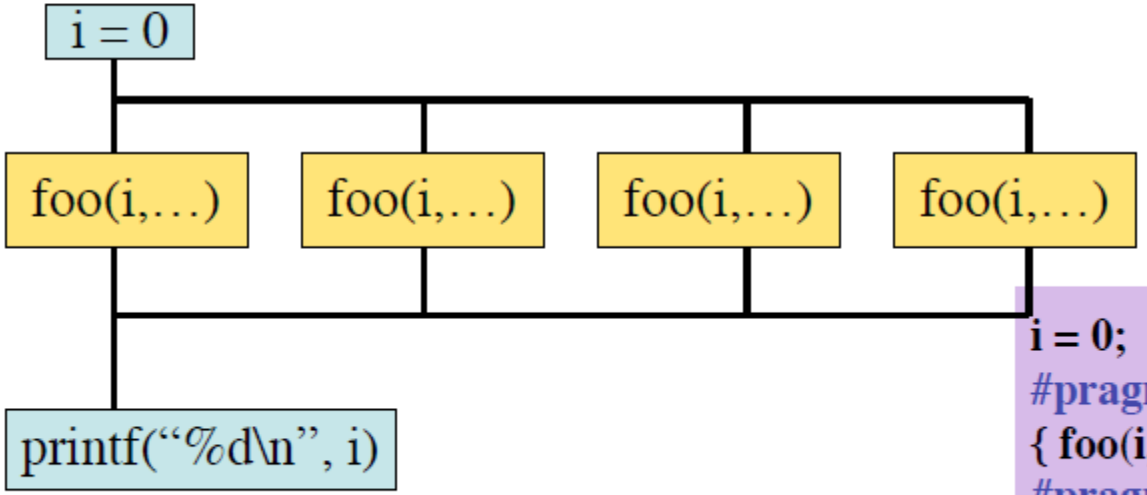
Variables REDUCTION

- ✓ Son variables que se utilizan en operaciones colectivas sobre elementos de un array

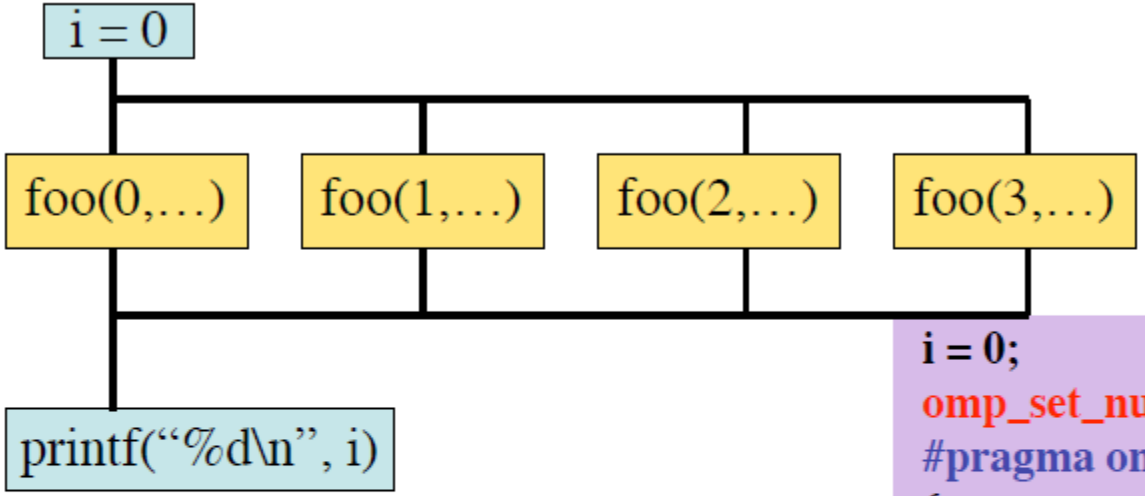
```
ASUM  = 0.0;
APROD = 1.0;
#pragma omp parallel for REDUCTION(+:ASUM)
for (i = 0; i < n; i++) {
   ASUM  = ASUM + A[i];
   APROD = APROD * A[i];
}
```

```
#pragma omp parallel
{
  /* Código a ser ejecutado por cada thread */
}
```

Código paralelo




```
i = 0;
#pragma omp parallel
{ foo(i,a,b,c)}
#pragma omp end parallel
printf("%d\n", i);
```




```
i = 0;
omp_set_num_threads(4);
#pragma omp parallel
{
   i = omp_thread_num();
    foo(i,a,b,c);
}
#pragma omp end parallel
printf("%d\n", i);
```

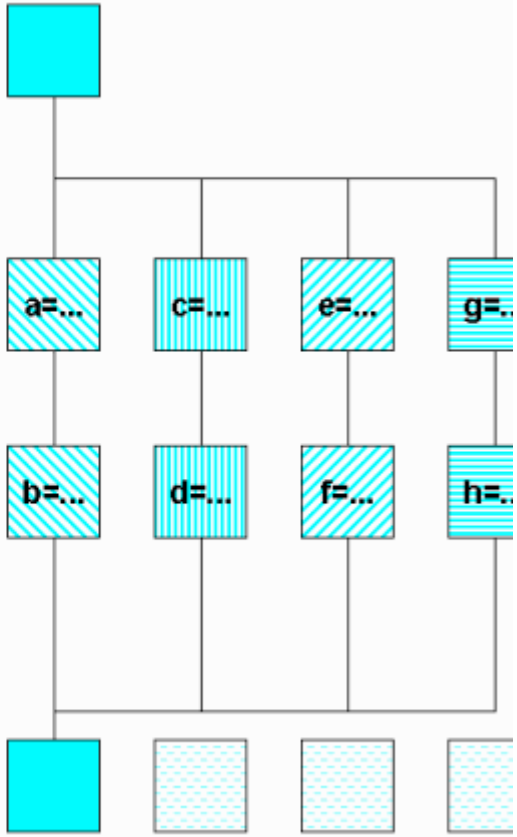
OpenMP runtime library

- OMP_GET_NUM_THREADS() devuelve el número de threads en ejecución
- OMP_GET_THREAD_NUM() devuelve el identificador de ese thread
- OMP_SET_NUM_THREADS(n) establece el número de threads
- ….

OpenMP sections Directives

C / C++:

```
#pragma omp parallel
{
#pragma omp sections
 {{  a=...;
     b=...; }
#pragma omp section
   {  c=...;
      d=...; }
#pragma omp section
   {  e=...;
      f=...; }
#pragma omp section
   {  g=...;
      h=...; }
 } /*omp end sections*/
} /*omp end parallel*/
```



Directiva Master

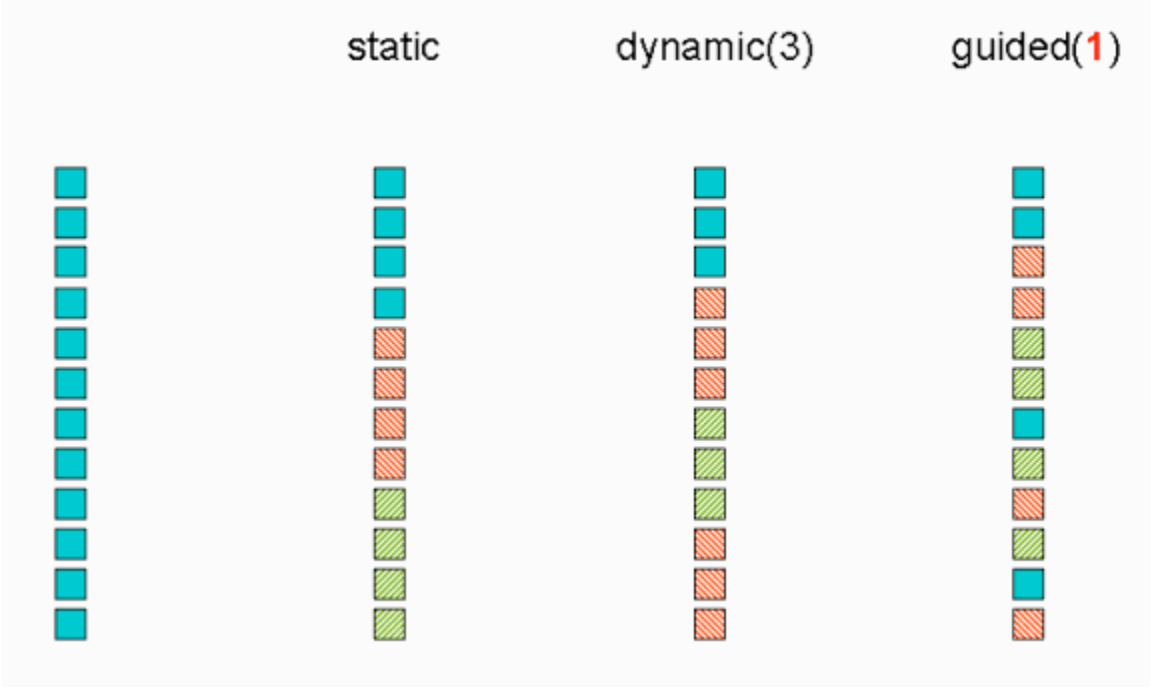
- ✓ La construcción master delimita un bloque estructurado que solo es ejecutado por el thread maestro. Los otros *threads* no lo ejecutan.

```
#pragma omp parallel private (tmp)
{
   acciones();
   #pragma omp master
   { acciones_maestro(); }
   #pragma barrier
   acciones();
}
```

Planificación de Tareas. Schedule

- ✓ Diferentes formas de asignar iteraciones a threads:
  - schedule(static [,chunk]) : “chunk” iteraciones se asignan de manera estática a los threads en round-robin
  - schedule (dynamic [,chunk]) Cada thread toma “chunk” iteraciones cada vez que está sin trabajo
  - schedule (guided [,chunk]) Cada thread toma iteraciones dinámicamente y progresivamente va tomando menos iteraciones.

Planificación de un loop

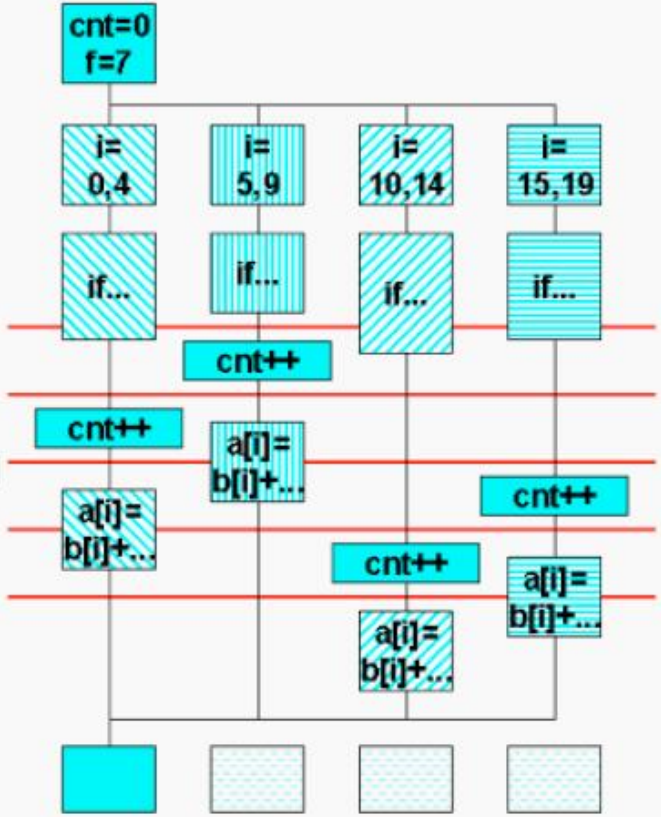


Exclusión Mutua. Sección crítica

```
#pragma omp parallel shared(x,y)
                    …
#pragma omp critical (section1)
    actualiza(x);
#pragma omp end critical(section1)
                    …
#pragma omp critical(section2)
    actualiza(y);
#pragma omp end critical(section2)
#pragma omp end parallel
```

Exclusión Mutua. Sección crítica

```
C++:   cnt = 0;
       f=7;
  #pragma omp parallel
  {
  #pragma omp for
       for (i=0; i<20; i++) {
         if (b[i] == 0) {
            #pragma omp critical
            cnt ++;
         } /* endif */
         a[i] = b[i] + f * (i+1);
       } /* end for */
  } /*omp end parallel */
```



Barreras

- ✓ Los *threads* se detienen hasta que todos alcancen la barrera

Sintaxis

```
#pragma omp barrier
```

Ejemplo:

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

bibliografía

- OpenMP Official Website: www.openmp.org
- OpenMP 4.0 Specifications
- Rohit Chandra, “Parallel Programming in OpenMP”. Morgan Kaufmann Publishers.
- The community of OpenMP researchers and developers in academia and industry: http://www.compunity.org/
- Conference papers: WOMPAT, EWOMP, WOMPEI, IWOMP
- http://www.nic.uoregon.edu/iwomp2005/index.html#program

GRACIAS