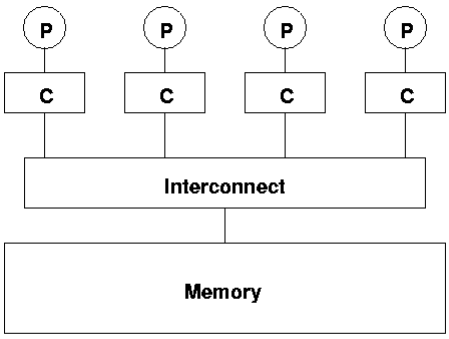
OpenMP

¿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++)

Shared Memory Systems (cont)

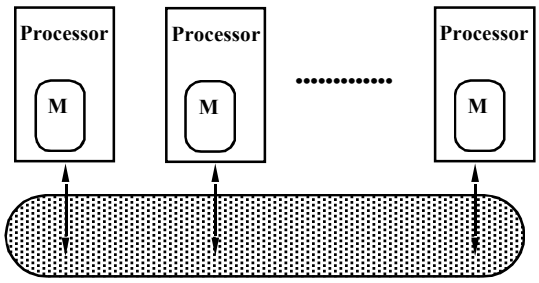


OpenMP

Pthreads

Shared memory

Distributed Memory Systems

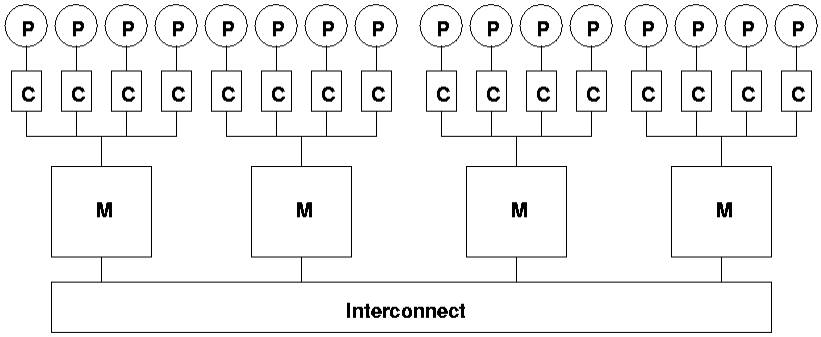


MPI

HPF

Interconnect

Clustered of SMPs



MPI

hybrid MPI + OpenMP

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]…]*
  - En Fortran, las directivas tienen la forma:
    - C$OMP *construct [clause [clause]…]*
    - !$OMP *construct [clause [clause]…]*
    - *$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

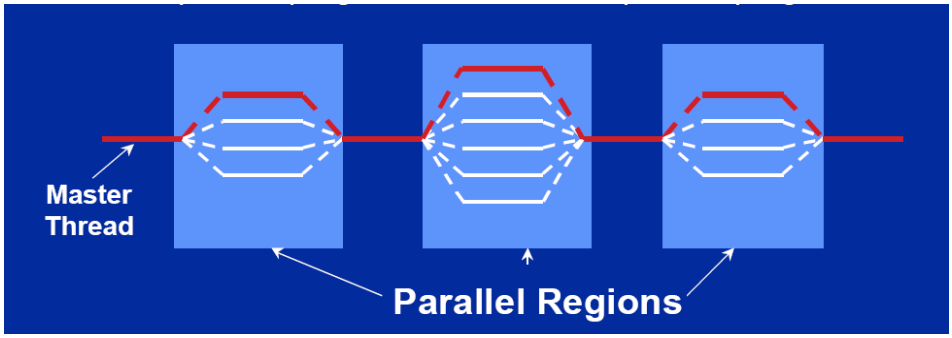
Programa Secuencial

```
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 en OpenMP



¿Cómo interactuan los threads?

- OpenMP es un modelo de memoria compartida.
  - Los threads se comunican utilizando variables compartidas.
- El uso inadecuado de variables compartidas origina carreras críticas:
- Para controlar las carreras críticas:
  - 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)
for (i = 0; i < n; i++) {
    a(i) = b(i) + c(i);
}
```

Alcance de los Datos
Ejemplo

```
#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];
}
```

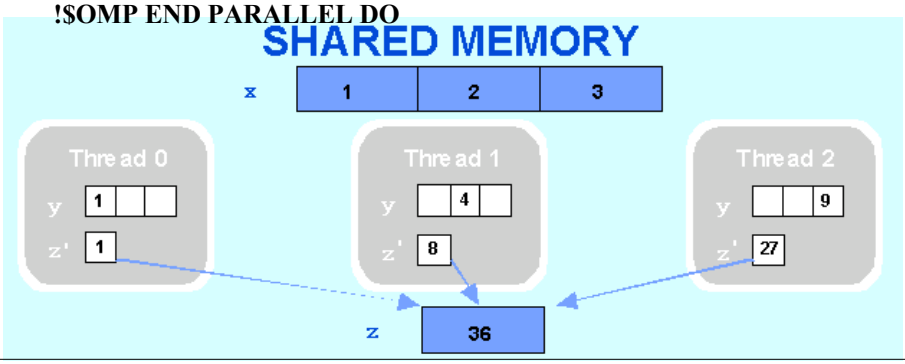
Variables REDUCTION

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];
}
```

Ejemplo

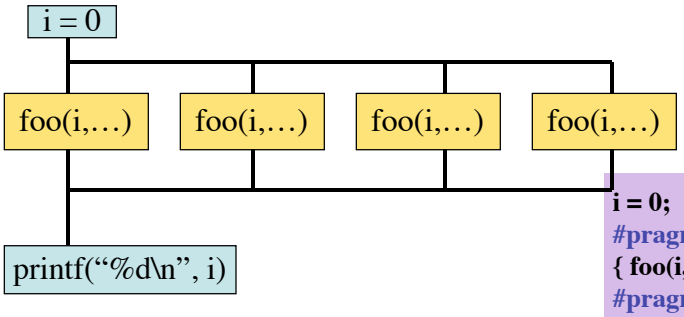
```
INTEGER x(3), y(3), z
!$OMP PARALLEL DO DEFAULT (PRIVATE), SHARED(x), &
!$OMP REDUCTION(+:z)
  DO k = 1, 3
    x(k) = k
    y(k) = k*k
    z = z + x(k) * y(k)
  END DO
!$OMP END PARALLEL DO
```



Regiones Paralelas

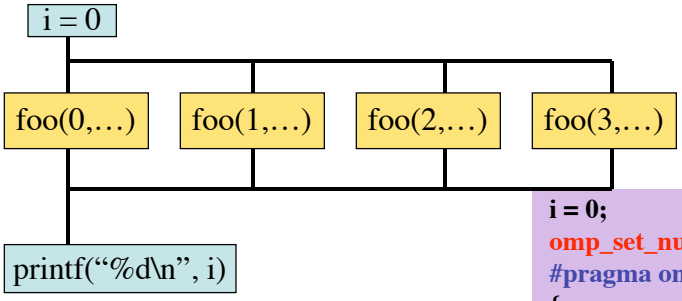
```
#pragma omp parallel
{
  /* Código a ser ejecutado por cada thread */
}
```

Código paralelo




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

Regiones Paralelas




```
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

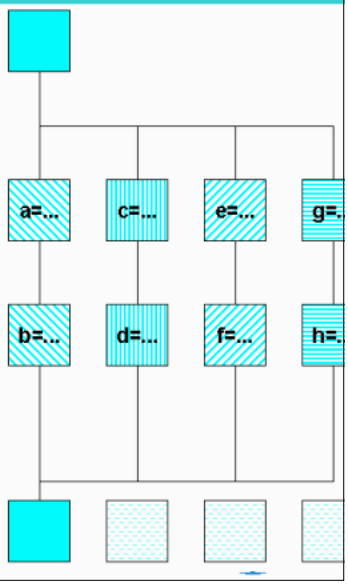
- `OMP_GET_NUM_THREADS()`
  regresa el número actual de threads
- `OMP_GET_THREAD_NUM()`
  regresa el identificador de ese thread
- `OMP_SET_NUM_THREADS(n)`
  indica el número de threads

etc…

OpenMP `sections` Directives (3)

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();
}
```

Compartir Trabajo Motivación

Código secuencial

```
for(i=0;i<n;i++) {a[i] = a[i] + b[i];}
```

Región paralela OpenMP

```
#pragma omp parallel
{
   int id, i, Nthreads, istart, iend;
   id = omp_get_thread_num(i);
   Nthreads = omp_get_num_threads();
   istart = id * N / Nthreads;
   iend = (id + 1) * N / Nthreads;
   for(i=istart;i<iend;i++) {a[i]=a[i]+b[i];}
}
```

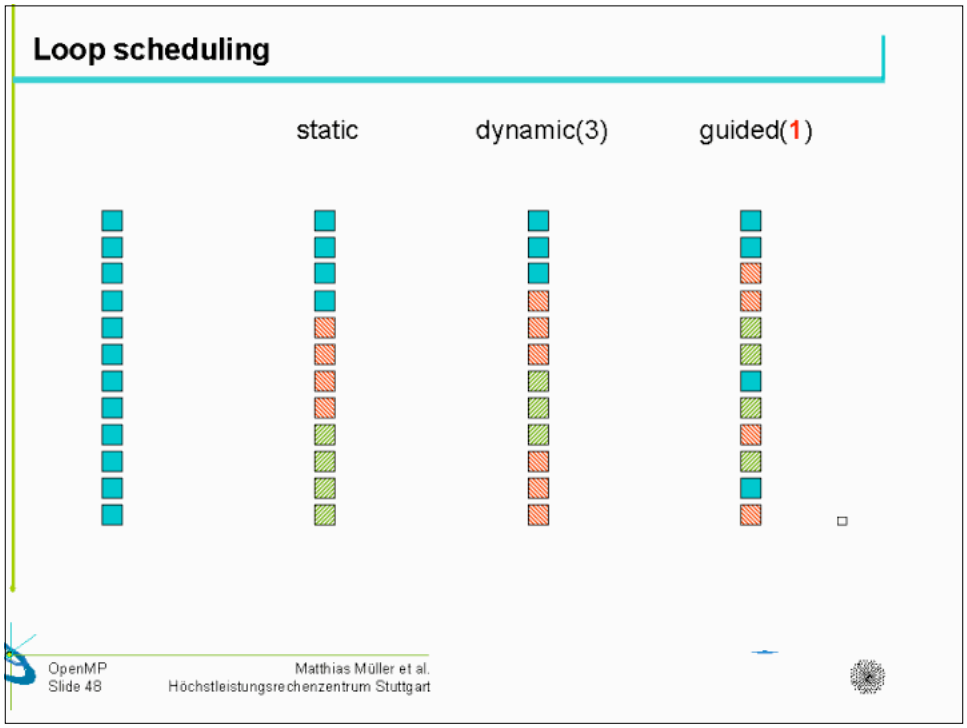
Región paralela y constructor para compartir Trabajo en OpenMP

```
#pragma omp parallel
#pragma omp for schedule(static)
   for(i=0;i<n;i++) {a[i] = a[i] + b[i];}
```

Planificación de Tareas
SCHEDULE

Direrentes 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.

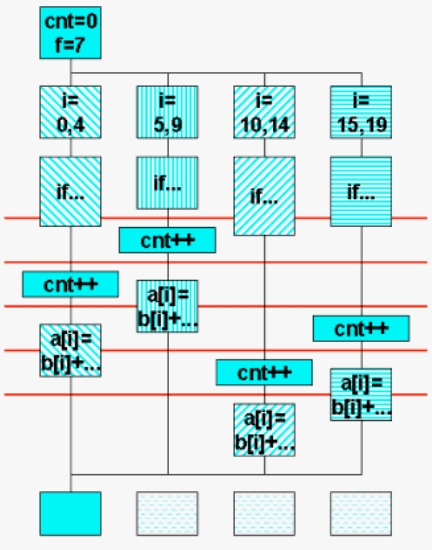


Exclusión Mutua
Sección Crítica

```
#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
```

penMP `critical` — an example (C/C++)

```
/ 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> }

Cálculo de PI Secuencial

```
static long num_steps = 100000;
double step;
void main ()
{     int i; double x, pi, sum = 0.0;
      step = 1.0/(double) num_steps;
      for (i=1;i<= num_steps; i++){
            x = (i-0.5)*step;
            sum = sum + 4.0/(1.0+x*x);
      }
      pi = step * sum;
}
```

Versión en OpenMP

```
#include <omp.h>
static long num_steps = 100000; double step;
#define NUM_THREADS 2
void main ()
{ int i; double x, pi, sum[NUM_THREADS];
   step = 1.0/(double) num_steps;
   omp_set_num_threads(NUM_THREADS);
   #pragma omp parallel
   { double x; int id;
      id = omp_get_thread_num();
      for (i=id, sum[id]=0.0;i< num_steps; i=i+NUM_THREADS){
         x = (i+0.5)*step;
         sum[id] += 4.0/(1.0+x*x);
   }
   }
   for(i=0, pi=0.0;i<NUM_THREADS;i++) pi += sum[i] * step;
}
```

OpenMP PI Program:
Work Sharing Construct

```
#include <omp.h>

static long num_steps = 100000; double step;

#define NUM_THREADS 2

void main ()

{ int i; double x, pi, sum[NUM_THREADS];
   step = 1.0/(double) num_steps;
   omp_set_num_threads(NUM_THREADS)
   #pragma omp parallel
   { double x; int id;
      id = omp_get_thread_num(); sum[id] = 0;
      #pragma omp for
      for (i=id;i< num_steps; i++){
       x = (i+0.5)*step;
       sum[id] += 4.0/(1.0+x*x);
      }
    }
    for(i=0, pi=0.0;i<NUM_THREADS;i++)pi += sum[i] * step;
}
```

Versión de PI que utiliza REDUCTION

```
#include <omp.h>
static long num_steps = 100000; double step;
#define NUM_THREADS 2
void main ()
{  int i; double x, pi, sum = 0.0;
   step = 1.0/(double) num_steps;
   omp_set_num_threads(NUM_THREADS);
   #pragma omp parallel for reduction(+:sum) private(x)
   for (i=1;i<= num_steps; i++){
        x = (i-0.5)*step;
        sum = sum + 4.0/(1.0+x*x);
   }
   pi = step * sum;
}
```

References

- OpenMP Official Website:
  - www.openmp.org
- OpenMP 2.5 Specifications
- An OpenMP book
  - Rohit Chandra, "Parallel Programming in OpenMP". Morgan Kaufmann Publishers.
- Compunity
  - The community of OpenMP re in academia and industry
  - http://www.compunity.org/
- Conference papers:
  - WOMPAT, EWOMP, WOMPEI, IWOMP
    - http://www.nic.uoregon.edu/iwomp2005/index.html#program