

Introducción a la Programación de Memoria Compartida .. con OpenMP®

Carlos Jaime BARRIOS HERNANDEZ, PhD.

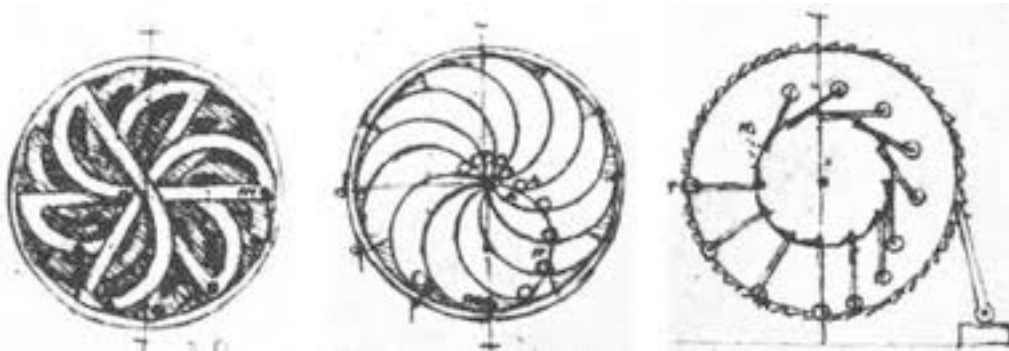
Escuela de Ingeniería de Sistemas e Informática

Universidad Industrial de Santander

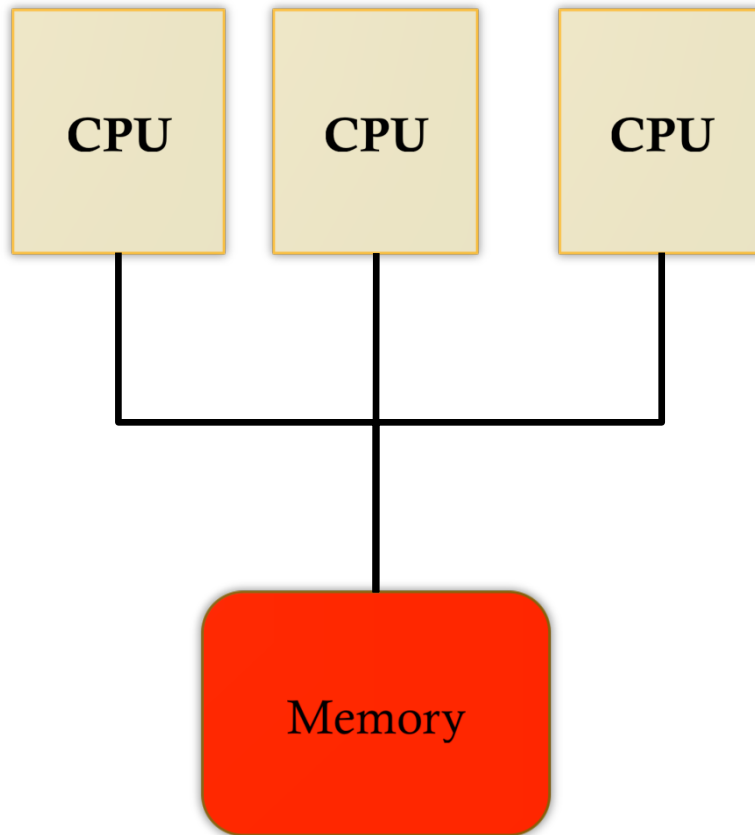
Las herramientas antes que el diseño



- OpenMP® como herramienta de aprendizaje para entender el modelo de programación de memoria compartida.
- A partir de la experiencia técnica pensar en el modelo teórico.



El Modelo de Memoria Compartida



Los Procesadores Interactúan con otro mediante variables compartida.

OpenMP es un estándar para programación de memoria compartida

OpenMP®

OpenMP es una especificación para implementaciones portables de paralelismo en FORTRAN y C/C++

La especificación contiene provee directivos de compilador para programación de multihilos, variables de ambiente y rutinas de biblioteca que controlan paralelismo

- Nivel de Cores
- Nivel de Procesadores

Soporta el modelo de paralelismo de datos

Paralelismo Incremental

Combina código serial y paralelo en un solo código fuente.



The OpenMP® API specification for parallel programming

www.openmp.org

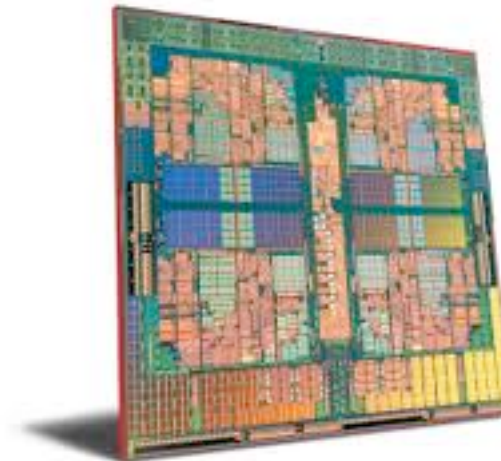
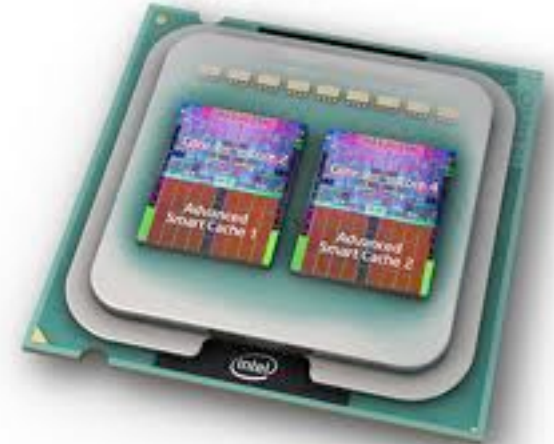
OpenMP es ideal para Arquitecturas Multi-procesador

Modelo de Hilos y de Memoria mapeado naturalmente

Ligero

Robusto

Disponible y usado para múltiples códigos sobre diferentes tecnologías disponibles (Intel, AMD)



Modelo de Memoria de OpenMP

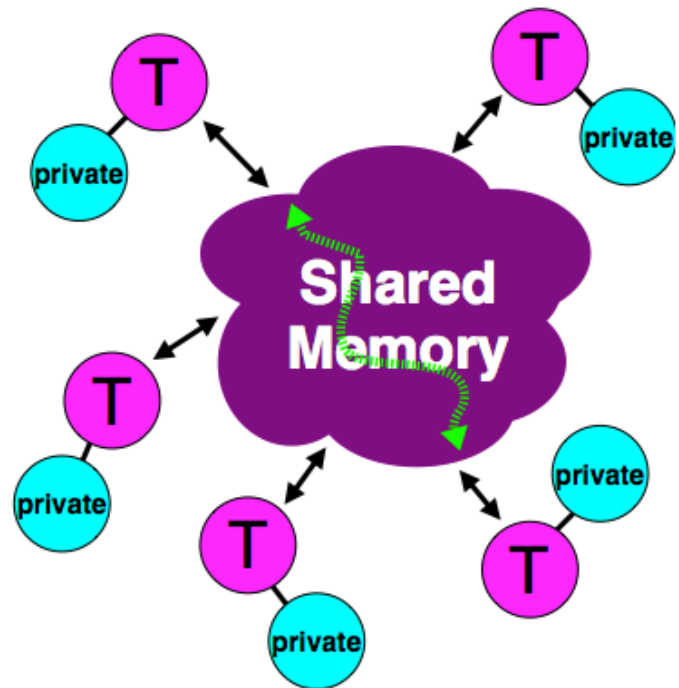
Todos los hilos tienen acceso a la misma memoria global compartida

Los datos pueden ser públicos o privados

Datos privados pueden ser accedidos únicamente por su propio hilo

Transferencia de Datos transparente al programador

Sincronización es implícita



Tomado de An Overview of OpenMP – SUN Microsystems

Arquitectura de OpenMP

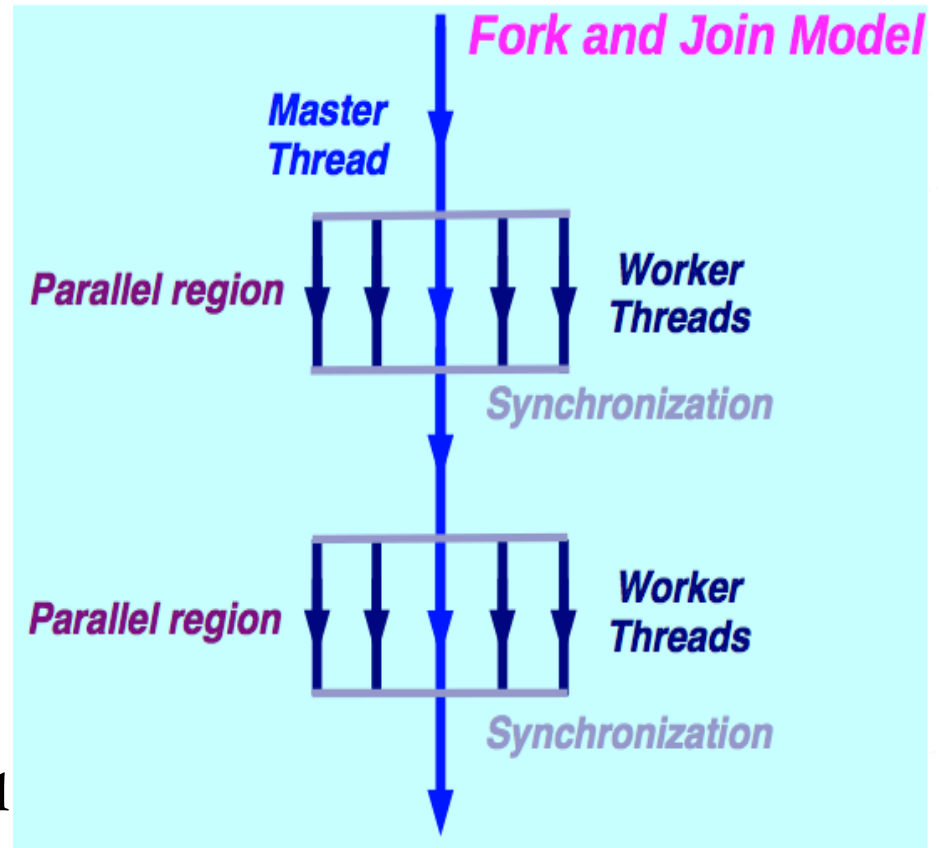
Modelo Fork-Join

Bloques de construcción para el trabajo en paralelo

Bloques de construcción para el ambiente de datos

Bloques de construcción para la sincronización

API extensiva para afinar el control



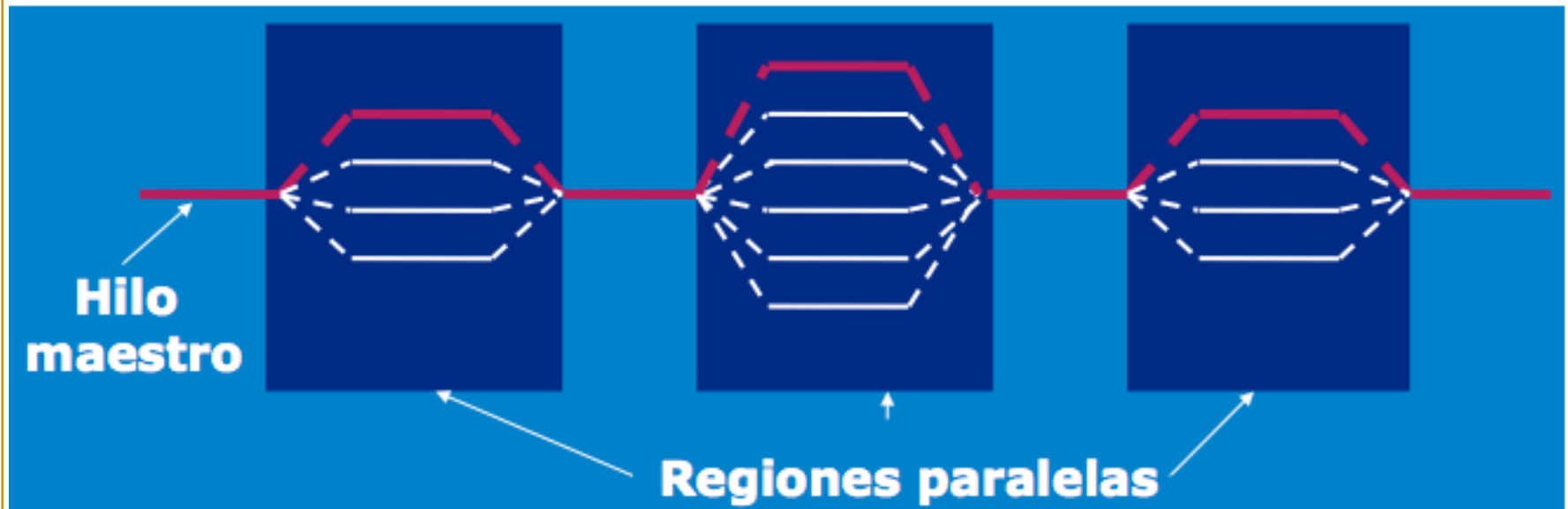
Modelo de Ejecución - Tomado de An Overview of OpenMP – SUN

Microsystems

Modelo de Programación

Paralelismo Fork-Join:

- El hilo maestro se divide en un conjunto de hilos como sea necesario
- El paralelismo se añade incrementalmente: el programa secuencial se convierte en un programa paralelo



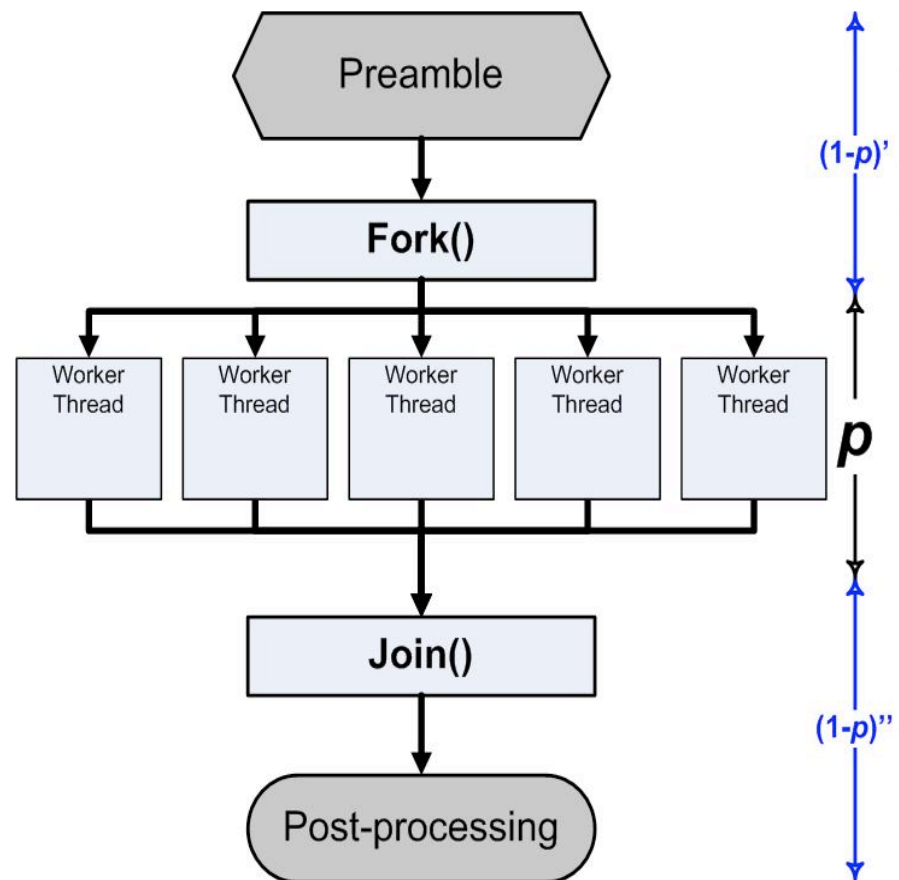
Modelo de Programación

Inicialmente solamente el hilo maestro está activo

El hilo maestro ejecuta el código secuencial

Fork: El hilo maestro crea hilos adicionales para ejecutar el código paralelo

Join: Al finalizar de código paralelo, los hilos creados mueren o se suspenden



Pragmas

Una pragma es un directivo al compilador.

La sintaxis es

```
#pragma omp <el resto de la pragma>
```

Ejemplo:

```
#pragma omp parallel for
```

es una directriz que dice al compilador que trate a
paralelizar el bucle for

La Pragma for Paralelo

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

El compilador tiene que verificar que cuando se ejecuta, habrá disponible la información necesaria para llevar a cabo las iteraciones

La sintaxis de la pragma for paralelo

for(indice = *primero*; indice $\geq$ $\left\{ \begin{array}{l} < \\ <= \\ >= \\ > \end{array} \right\}$ *ultimo*; $\left\{ \begin{array}{l} \text{indice} ++ \\ ++ \text{indice} \\ \text{indice} -- \\ -- \text{indice} \\ \text{indice} + = \text{inc} \\ \text{indice} - = \text{inc} \\ \text{indice} = \text{indice} + \text{inc} \\ \text{indice} = \text{inc} + \text{indice} \\ \text{index} = \text{indice} - \text{inc} \end{array} \right\}$

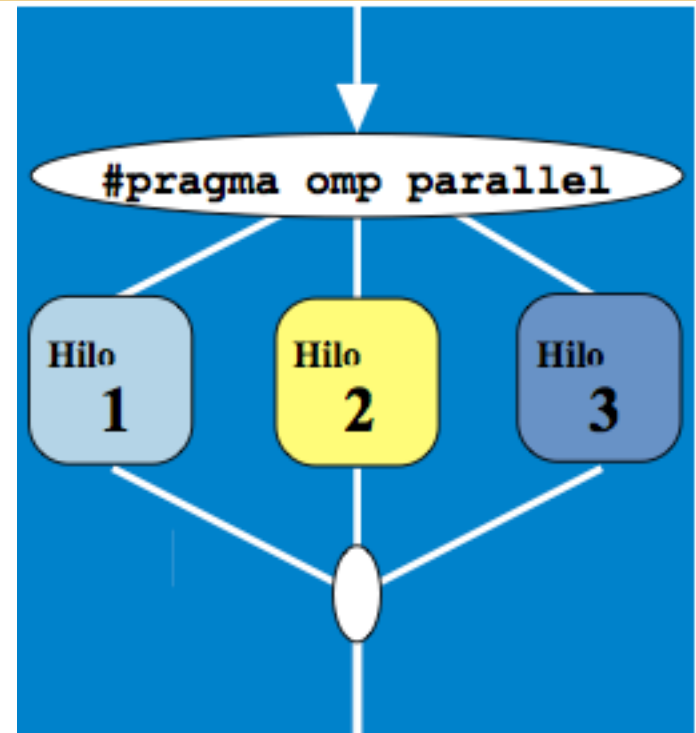
Regiones Paralelas

Define una región paralela sobre un bloque de código estructurado

Los hilos son creados como “parallel”

Los hilos se bloquean al final de la región

Los datos se comparten entre hilos al menos que se especifique otra cosa



Variables Compartidas y Variables Privadas

Una variable compartida tiene la misma dirección en el contexto de ejecución de cada hilo.

Una variable privada tiene una dirección distinta en el contexto de ejecución de cada hilo.

Un hilo no puede acceder las variables privadas de otro hilo.

OMP_NUM_THREADS

Establece el numero de hilos usando una variable de ambiente

No hay un estándar por defecto en esta variable

- # de hilos = # de procesadores = # Cores
- Los compiladores INTEL usan esto por defecto

```
set OMP_NUM_THREADS=4
```

Función

omp_set_num_threads

Se usa para asignar el número de hilos a ser activos en secciones paralelas del código

Se puede llamar en varios puntos del programa

```
void omp_set_num_threads (int  
t)
```

omp_get_thread_num()

Todo hilo tiene una identificación que es el número del hilo

omp_get_thread_num()
devuelve el número del hilo

Función `omp_get_num_procs`

Devuelve el número de procesadores físicos que están disponibles para el uso del programa paralelo

```
int omp_get_num_procs (void)
```

Hello World

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

int main (int argc, char *argv[]) {
    int p,th_id;
    p=omp_get_num_procs();
    omp_set_num_threads(p);
    #pragma omp parallel private(th_id);
    {
        th_id = omp_get_thread_num();
        printf("Hello World from thread %d\n", th_id);
    }
    return 0;
}
```

Para compilar y ejecutar

Compilar: `cc -openmp -o hello.c hello`

Ejecutar: Hay que especificar el número de hilos

Fuera del programa con

`setenv OMP_NUM_THREADS = número de hilos`

Dentro del programa con

`omp_set_num_threads( número de hilos )`

Otro Ejemplo

For-Loop con iteraciones independientes

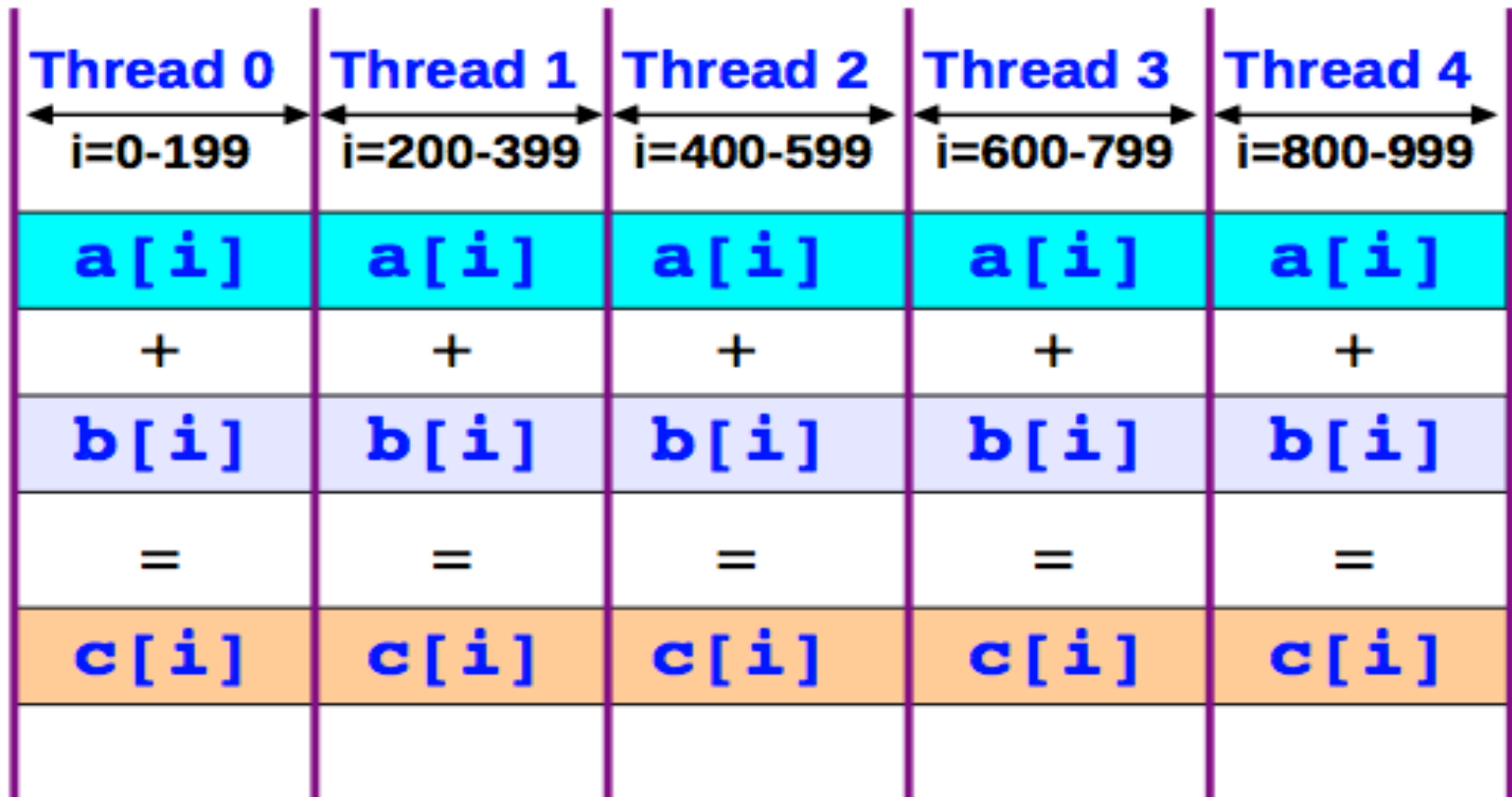
```
for (int i=0; i<n; i++)  
    c[i] = a[i] + b[i];
```

For-Loop paralelizado usando un pragma de OpenMP

```
#pragma omp parallel for  
for (int i=0; i<n; i++)  
    c[i] = a[i] + b[i];
```

```
cc -fopenmp source.c -o source  
Setenv OMP_NUM_THREADS 5  
source.out
```

Ejemplo de Ejecución Paralela



Directivas

Ambiente de Ejecución

Variables de Ambiente

Región Paralela

Numero de Hilos

Numero de Hilos

Trabajo Compartido
(Worksharing)

ID del Hilo

Tipo de Calendarizador

Atributos de Datos
Compartidos : private,
firstprivate, lastprivate,
shared, reduction

Ajuste Dinamico de
Hilos

Ajuste Dinamico de
Hilos

Orfandad (Oprhaning)

Paralelismo Anidado

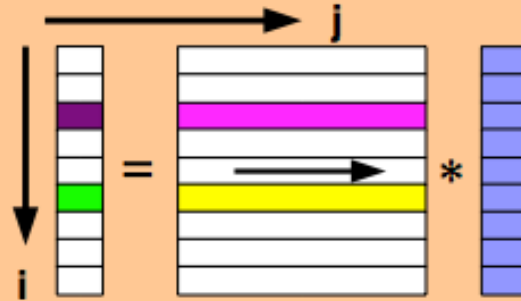
Paralelismo Anidado

Limitador del tiempo
de reloj – Temporizador

Bloqueo

Ejemplo: Matrix -Vector

```
#pragma omp parallel for default(none) \
                    private(i,j,sum) shared(m,n,a,b,c)
for (i=0; i<m; i++)
{
    sum = 0.0;
    for (j=0; j<n; j++)
        sum += b[i][j]*c[j];
    a[i] = sum;
}
```



TID = 0

for (i=0,1,2,3,4)

i = 0

sum = $\sum$ b[i=0][j]*c[j]

a[0] = sum

i = 1

sum = $\sum$ b[i=1][j]*c[j]

a[1] = sum

TID = 1

for (i=5,6,7,8,9)

i = 5

sum = $\sum$ b[i=5][j]*c[j]

a[5] = sum

i = 6

sum = $\sum$ b[i=6][j]*c[j]

a[6] = sum

... etc ...

```
#pragma omp parallel if (n>limit) default(none) \
    shared(n,a,b,c,x,y,z) private(f,i,scale)
{
```

```
    f = 1.0;
```

```
#pragma omp for nowait
```

```
    for (i=0; i<n; i++)
        z[i] = x[i] + y[i];
```

```
#pragma omp for nowait
```

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

```
#pragma omp barrier
```

```
    ....
    scale = sum(a,0,n) + sum(z,0,n) + f;
    ....
```

```
} /*-- End of parallel region --*/
```

Statement is executed
by all threads

parallel loop
(work is distributed)

parallel loop
(work is distributed)

synchronization

Statement is executed
by all threads

parallel region

Evaluacion de Rendimiento en OpenMP

Ejemplo: Medir Tiempos

```
double empezar,terminar;
```

```
empezar=omp_get_wtime( );
```

```
... algun código
```

```
terminar=omp_get_wtime();
```

```
printf(“TIEMPO=%lf\n”,empezar-terminar)
```

El resultado es en segundos.

Paralelismo Funcional

OpenMP nos permite asignar hilos distintos a partes distintas del código (paralelismo funcional)

$$y_i^{(1)} = \frac{y_{i+1} - y_{i-1}}{2d} - \frac{2d^3}{2d3!} y_i^{(3)} + \dots;$$

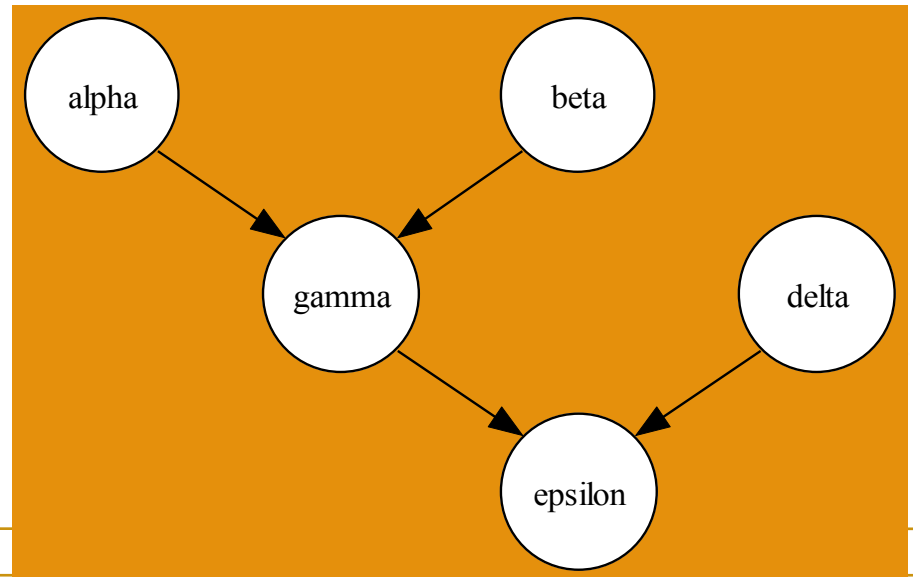
$$y_i^{(1)} = \frac{y_{i+1} - y_i}{d} - \frac{d^2}{2!d} y_i^{(2)} + \dots;$$

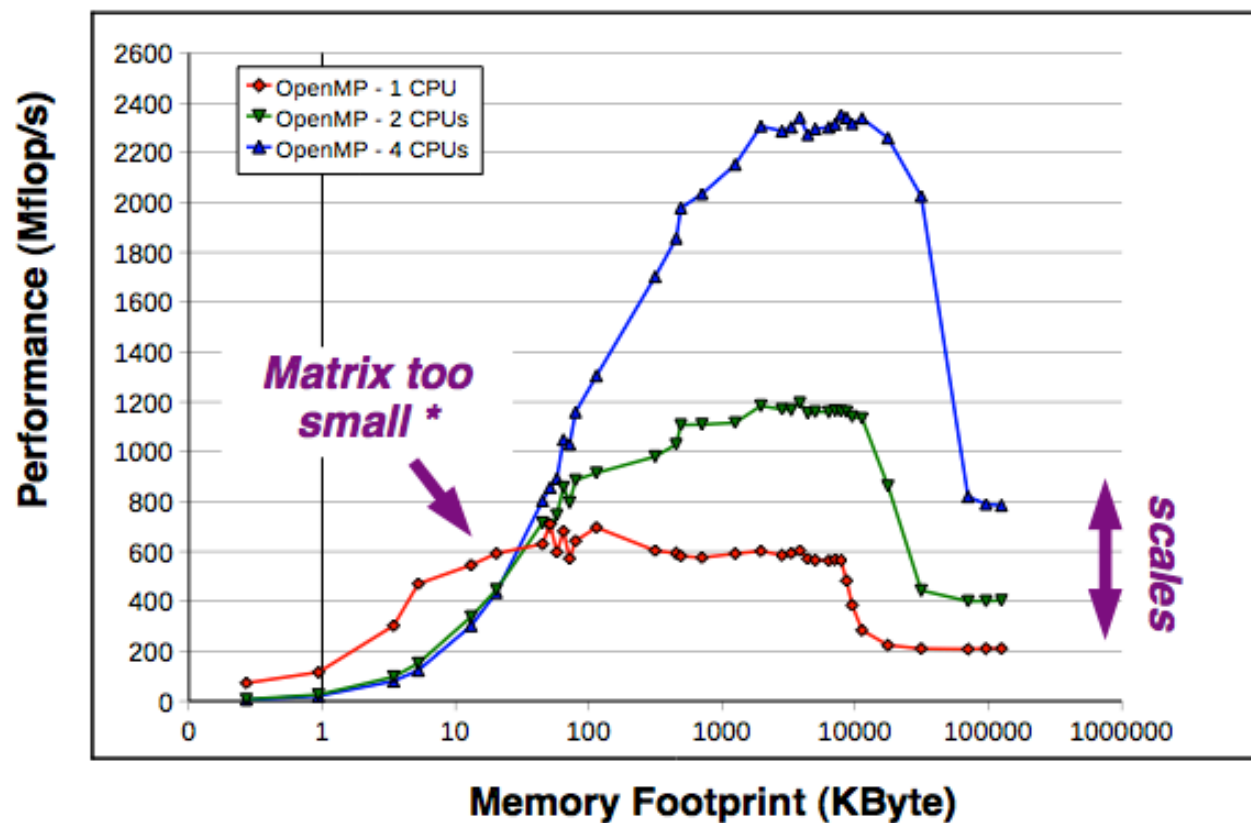
$$y_i^{(1)} = \frac{y_i - y_{i-1}}{d} + \frac{d^2}{2!d} y_i^{(2)} + \dots$$

Ejemplo de Paralelismo Funcional

```
v = alpha() ;  
w = beta() ;  
x = gamma(v, w) ;  
y = delta() ;  
printf ("%6.2f\n", epsilon(x,y)) ;
```

Se puede ejecutar
alpha, beta, and delta
en paralelo.





**) With the IF-clause in OpenMP this performance degradation can be avoided*

Conclusiones

OpenMP permite explotar no solo paralelismo de datos, sino también de tareas.

OpenMP debe considerarse cuando:

- Se vayan a programar cores y es no es fácilmente la determinación la granularidad del problema, pero si el uso de memoria compartida.

La decisión de cuantos hilos usar o no y de cómo sincronizarlos es tomada por el compilador y no necesariamente por el programador.

Lecturas Recomendadas

Parallel Scalability Isn't Child's Play, Mark B. Friedman

- Parte 1:
<http://blogs.msdn.com/b/ddperf/archive/2009/03/16/parallel-scalability-isn-t-child-s-play.aspx>
- Parte 2:
<http://blogs.msdn.com/b/ddperf/archive/2009/04/29/parallel-scalability-isn-t-child-s-play-part-2-amdahl-s-law-vs-gunther-s-law.aspx>

An Introduction to OpenMP, Robinson Rivas, SC-CAMP 2011
http://www.sc-camp.org/_pdf/OpenMP%20-%20sccamp%202011.pdf

OpenMP® Official Site: <http://www.openmp.org>

OpenMP Tutorial at Lawrence Livermore National Laboratory:
<https://computing.llnl.gov/tutorials/openMP/>