INTRODUCCIÓN A LA PROGRAMACIÓN DE MEMORIA COMPARTIDA
.. CON OPENMP®

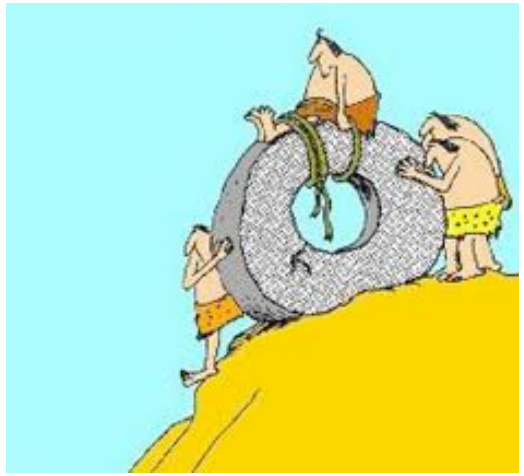
Carlos Jaime BARRIOS HERNANDEZ, PhD.

@carlosjaimebh

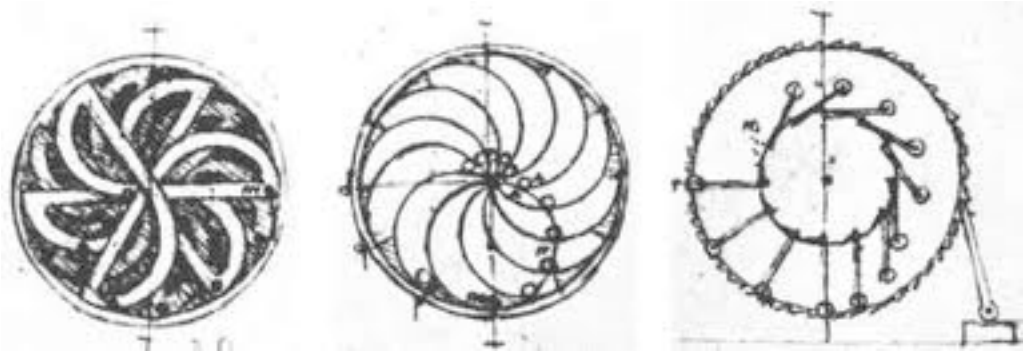
Escuela de Ingeniería de Sistemas e Informática

Universidad Industrial de Santander

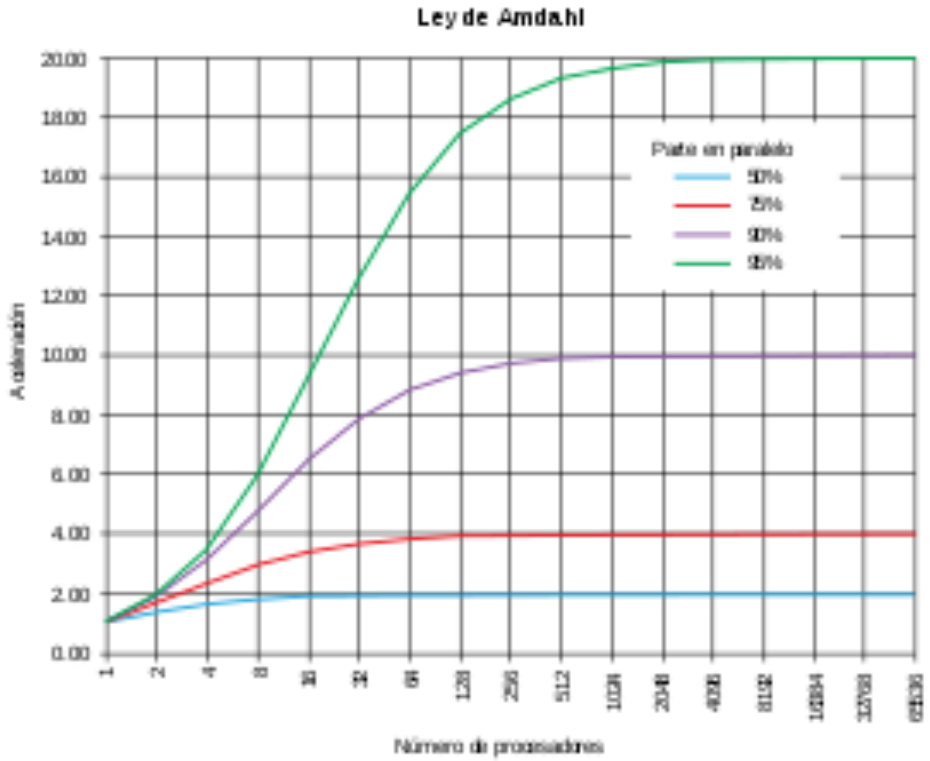
LAS HERRAMIENTAS ANTES QUE EL DISEÑO



- 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.



PERO ANTES, LA LEY DE AMHDAL



La mejora obtenida en el rendimiento de un sistema debido a la alteración de uno de sus componentes está limitada por la fracción de tiempo que se utiliza dicho componente.

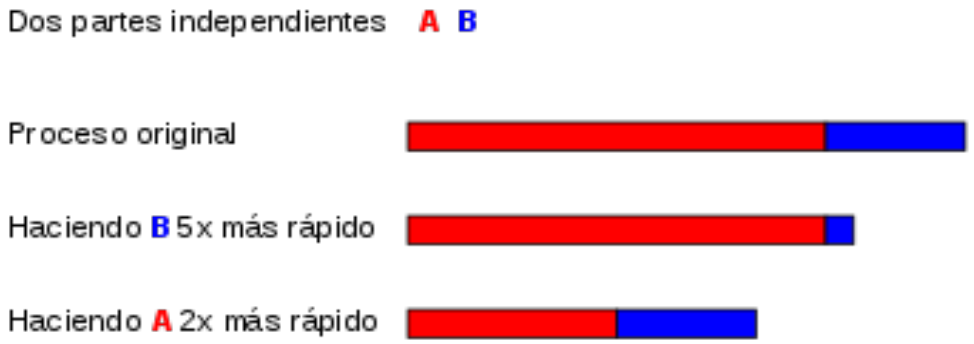
$$T_m = T_a \cdot \left((1 - F_m) + \frac{F_m}{A_m} \right)$$

F_m = fracción de tiempo que el sistema utiliza el subsistema mejorado
A_m = factor de mejora que se ha introducido en el subsistema mejorado.
T_a= tiempo de ejecución antiguo.
T_m= tiempo de ejecución mejorado.

Gene Amdahl, "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities", AFIPS Conference Proceedings, (30), pp. 483-485, 1967.

https://es.wikipedia.org/wiki/Ley_de_Amdahl

ACELERACION DESDE AMHDAL



Asumiendo que una tarea tiene dos partes independientes, **A** y **B**, consumiendo **B** el 25% del tiempo total de computación. Trabajando muy duro se puede realizar **B** 5 veces más rápido y sin embargo esto sólo reduce el tiempo de computación un poco; en contraste, una pequeña mejora de **A** hace que ésta vaya el doble de rápido. Esto hace que sea mucho mejor la optimización de **A** que de **B** aunque se mejore mucho más **B** (5x contra 2x).

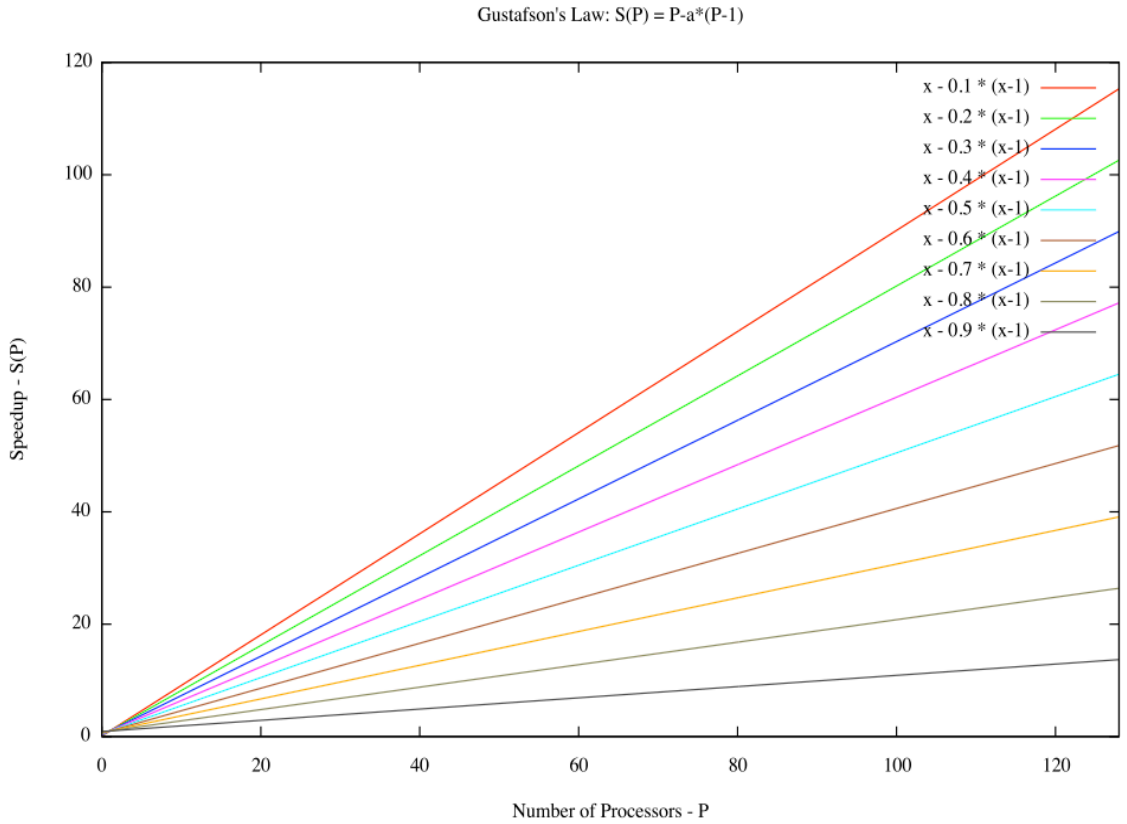
https://es.wikipedia.org/wiki/Ley_de_Amdahl

$$A = \frac{1}{(1 - F_m) + \frac{F_m}{A_m}}$$

Donde
A es la aceleración o ganancia en velocidad conseguida en el sistema completo debido a la mejora de uno de sus subsistemas.
A_m es el factor de mejora que se ha introducido en el subsistema mejorado.
F_m es la fracción de tiempo que el sistema utiliza el subsistema mejorado.
En la bibliografía en inglés A es representado como S.

¿Que pasa entonces si consideramos la cantidad de procesadores?

LEY DE GUSTAFON-BARSIS



https://es.wikipedia.org/wiki/Ley_de_Gustafson

- La Ley de Gustafon-Barsis establece que cualquier problema suficientemente grande puede ser eficientemente paralelizado.

$$S(P) = P - \alpha \cdot (P - 1)$$

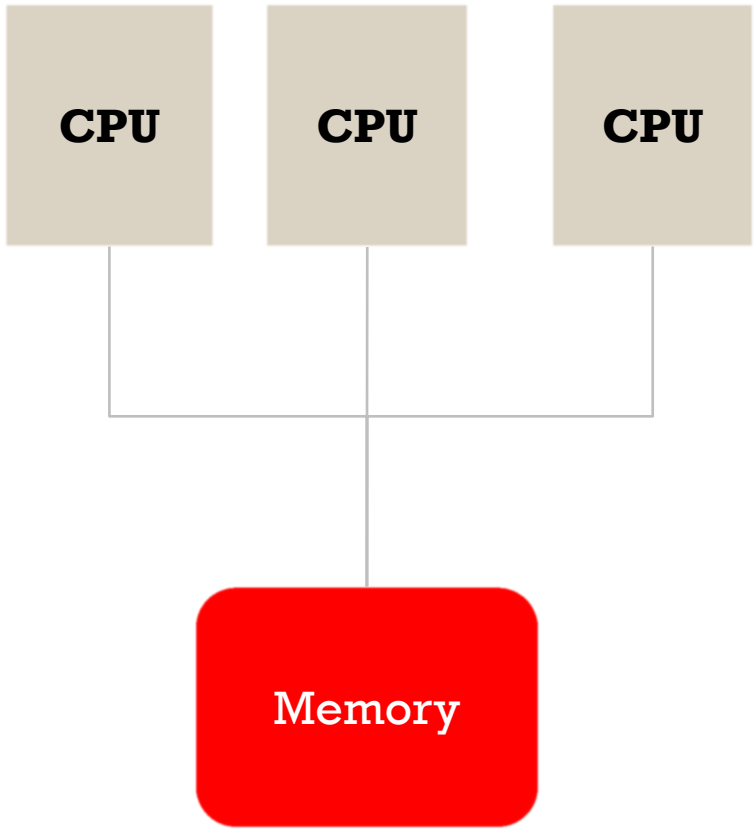
donde P es el número de procesadores, S es el speedup, y α la parte no paralelizable del proceso.

Limitaciones:

- Algunos problemas no son tan grandes.
- Los algoritmos no lineales dificultan la utilización de las ventajas del paralelismo.
- A nivel de núcleo o hilo, es necesario observar la aceleración secuencial.

Reevaluating Amdahl's Law, John L. Gustafson, Communications of the ACM, volumen 31 (mayo 1988)
Amdahl's Law in the Multicore Era, Mark D. Hill and Michael R. Marty, IEEE Computer, vol. 41, pp. 33–38, July 2008. Also UW CS-TR-2007-1593, April 2007.
Extending Amdahl's Law for Energy-Efficient Computing in the Many-Core Era ., Dong Hyuk Woo and Hsien-Hsin S. Lee, IEEE Computer, vol. 41, No. 12, pp.24-31, December 2008.

EL MODELO DE MEMORIA COMPARTIDA



- Los Procesadores Interactúan con otro mediante variables compartida.
- OpenMP es un estándar para programación de memoria compartida

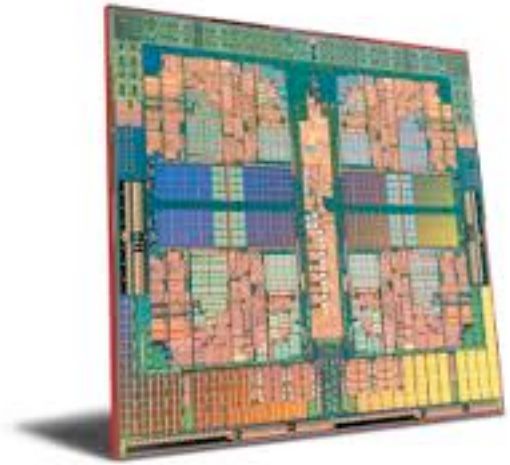
OPENMP®

- OpenMP es una especificación para implementaciones portables de paralelismo en FORTRAN y C/C++
- Las 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.
- **La especificación actual es la 5.1**

The OpenMP® API specification for parallel programming
www.openmp.org

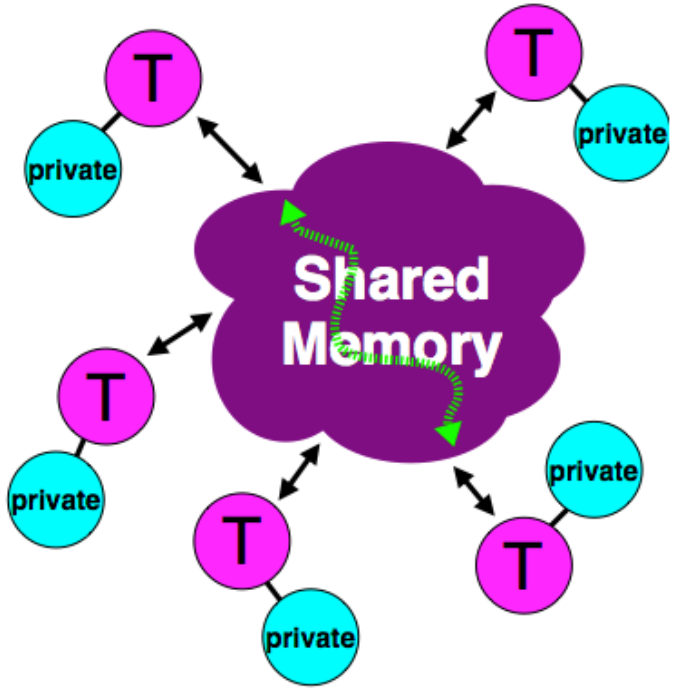
OPENMP ES IDEAL PARA ARQUITECTURAS MULTICORE

- 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

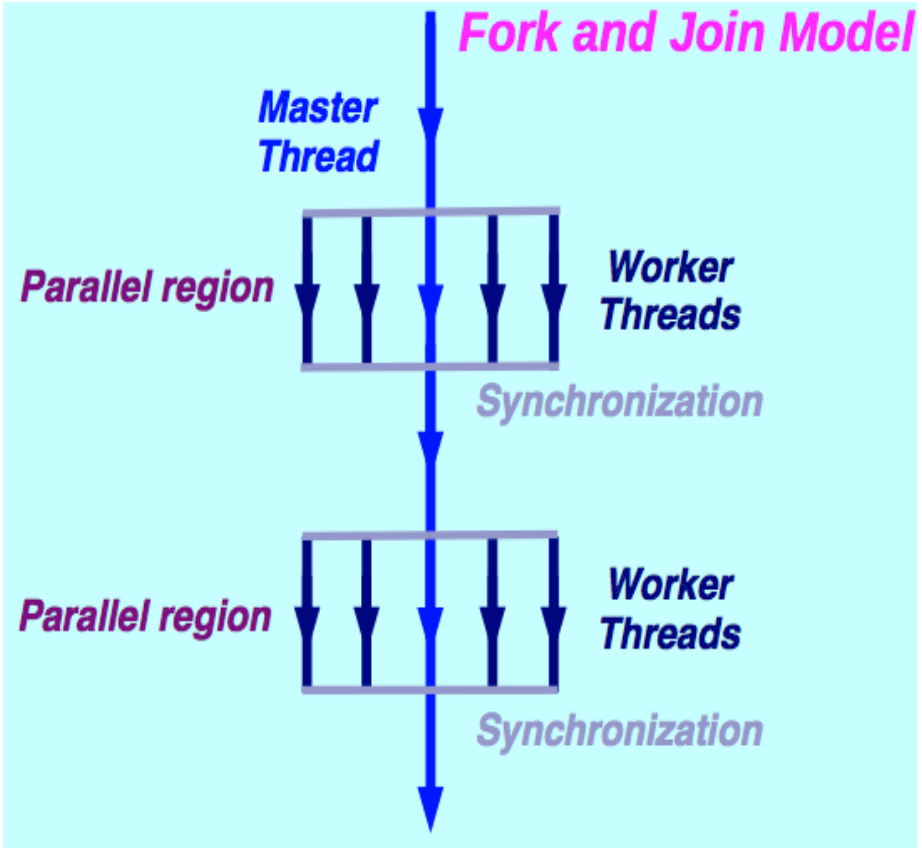
- Todos los hilos tienen acceso a la misma memoria global compartida
- Los datos pueden ser públicos o privados
- Datos privados pueden ser accedidos únicamente por su propio hilo
- Transferencia de Datos transparente al programador
- Sincronización es implícita



Tomado de An Overview of OpenMP – SUN Microsystems

ARQUITECTURA DE OPENMP

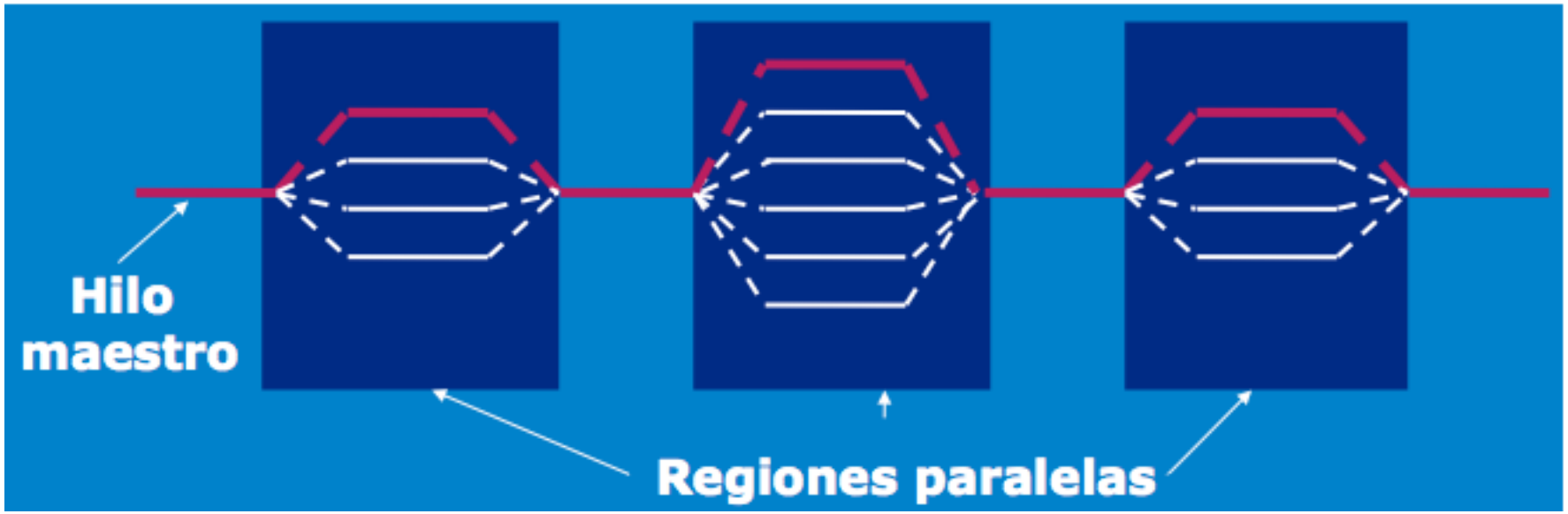
- Modelo Fork-Join
- Bloques de construcción para el trabajo en paralelo
- Bloques de construcción para el ambiente de datos
- Bloques de construcción para la sincronización
- API extensiva para afinar el control



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

MODELO DE PROGRAMACIÓN

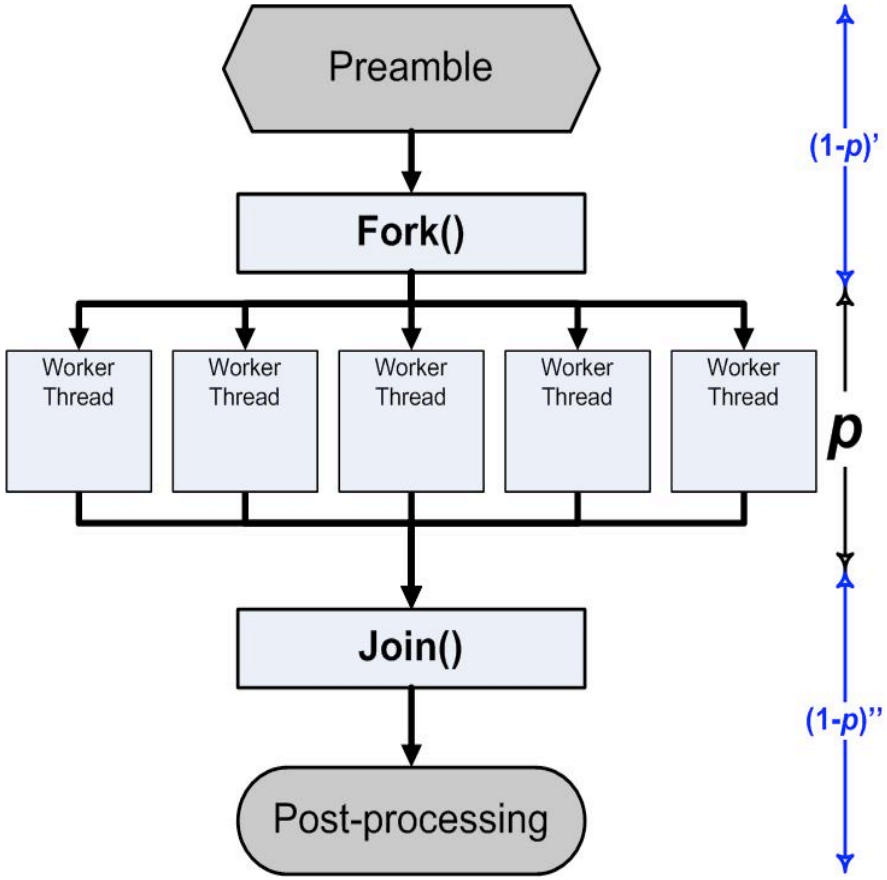
- Sigue el mismo Paralelismo Fork-Join:
  - El hilo maestro se divide en un conjunto de hilos como sea necesario
  - El paralelismo se añade incrementalmente: el programa secuencial se convierte en un programa paralelo



Tomado de Programación en OpenMP – Robinson Rivas SC-CAMP 2011

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



Parallel Scalability Isn't Child's Play, Part 2: Amdahl's Law vs. Gunther's Law

PRAGMAS

- Una pragma es un directivo al compilador.
- La sintaxis es

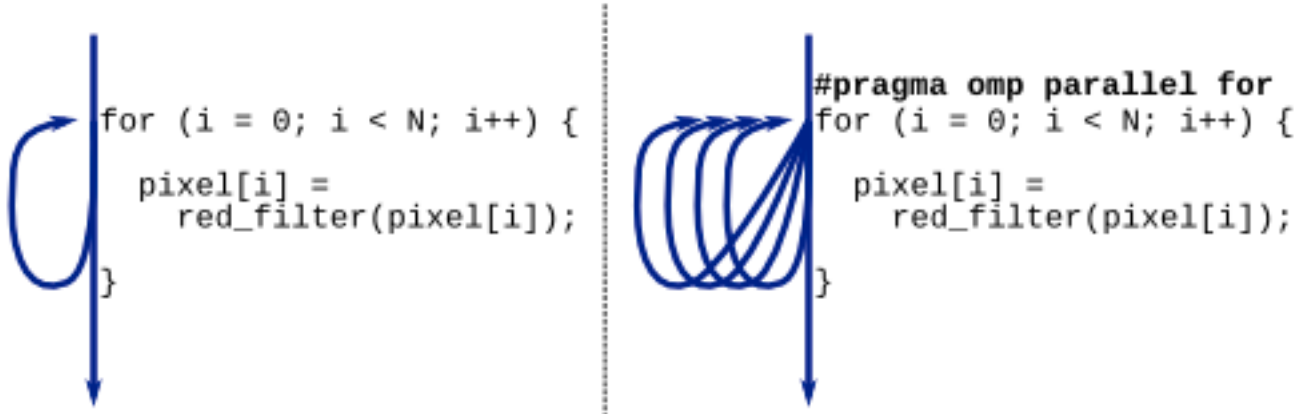
#pragma omp <el resto de la pragma>

Ejemplo:

#pragma omp parallel for

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

EJECUCIÓN



C

```c
int i;
int a[10] = {3,6,3,1,7,9,5,6,9,1};

for (i = 0; i < 10; i++) {
    a[i] *= 2;
}
```

C

```c
int i; int a[10] = {3,6,3,1,7,9,5,6,9,1};

#pragma omp parallel for
for (i = 0; i < 10; i++) {
    a[i] *= 2;
}
```

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

Tomado del Curso de Open MP de Doroty Bollman http://pegasus.uprm.edu/bollman/

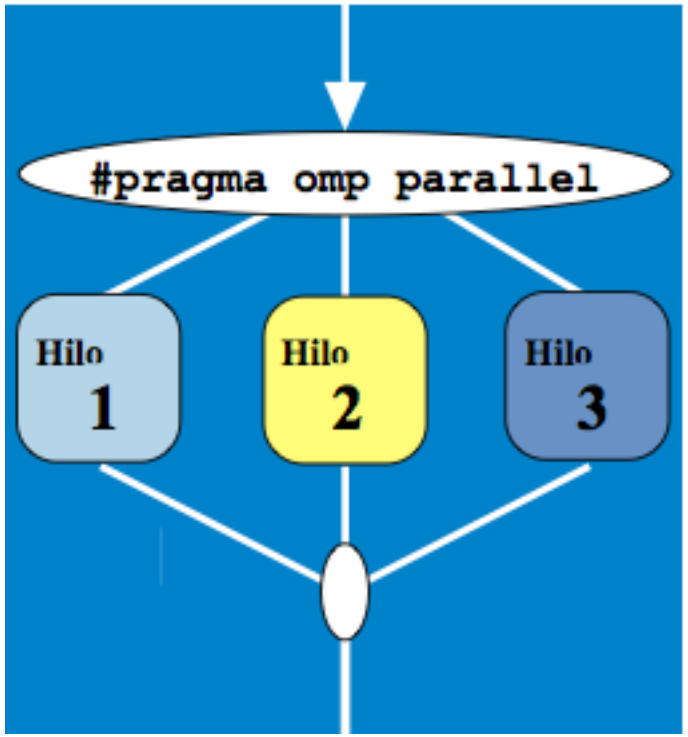
LA SINTAXIS DE LA PRAGMA FOR PARALELO

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

Tomado del Curso de Open MP de Doroty Bollman
http://pegasus.uprm.edu/bollman/

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



Tomado de Programación en OpenMP – Robinson Rivas SC-CAMP 2011

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 accesar 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 fisicos que están dispondible 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

Thread 0	Thread 1	Thread 2	Thread 3	Thread 4
i=0-199	i=200-399	i=400-599	i=600-799	i=800-999
a[i]	a[i]	a[i]	a[i]	a[i]
+	+	+	+	+
b[i]	b[i]	b[i]	b[i]	b[i]
=	=	=	=	=
c[i]	c[i]	c[i]	c[i]	c[i]

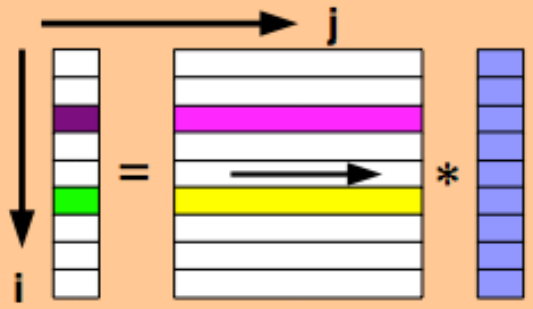
Ejemplo de Ejecución - Tomado de An Overview of OpenMP – SUN Microsystems

COMPONENTES DE OPENMP®

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	

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 = Σ  b[i=0][j]*c[j]
   a[0] = sum
 i = 1
sum = Σ  b[i=1][j]*c[j]
   a[1] = sum
```

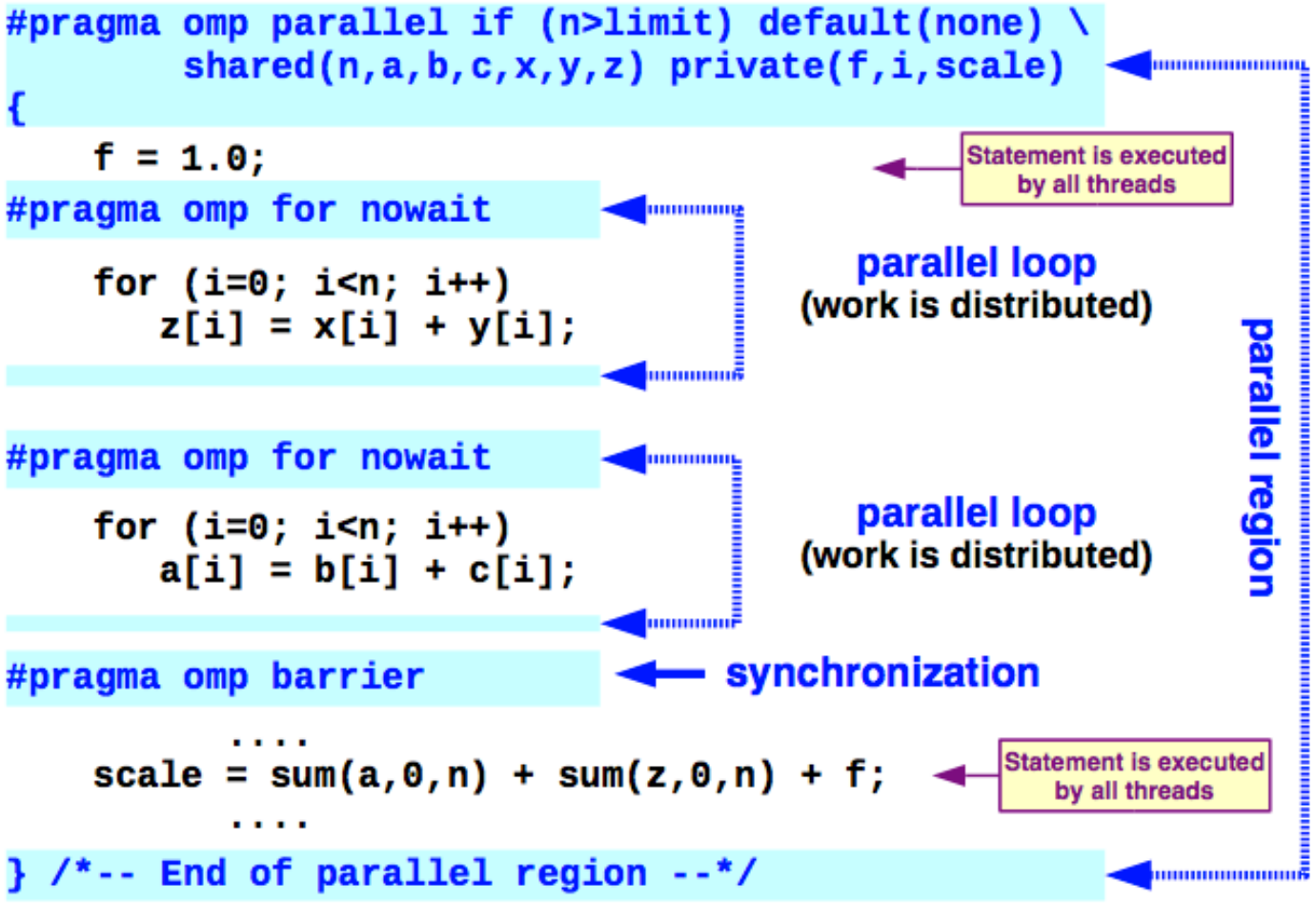
TID = 1

```
for (i=5,6,7,8,9)
 i = 5
sum = Σ  b[i=5][j]*c[j]
   a[5] = sum
 i = 6
sum = Σ  b[i=6][j]*c[j]
   a[6] = sum
```

... etc ...

Tomado de An Overview of OpenMP – SUN Microsystems

UN EJEMPLO MAS ELABORADO...



Tomado de An Overview of OpenMP – SUN Microsystems

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)

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

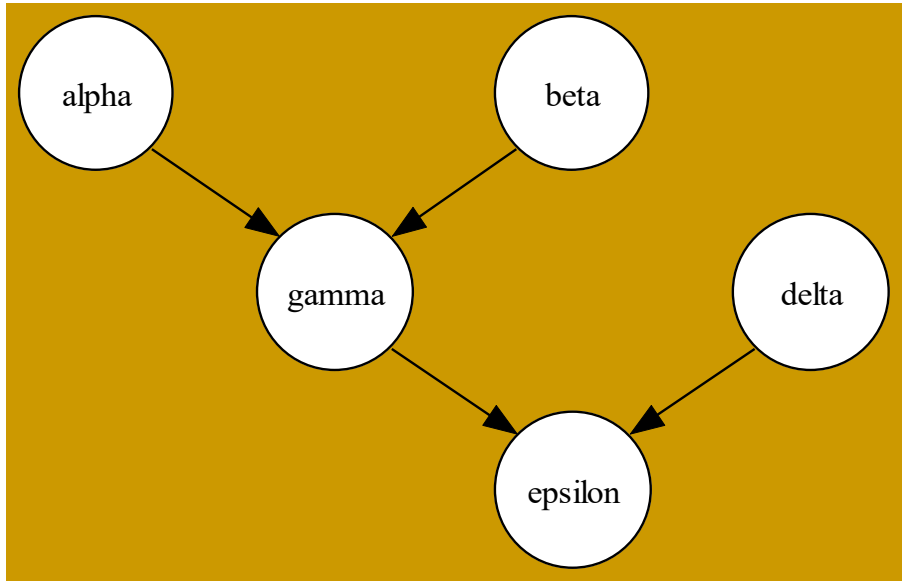
$$y_i^{(1)} = \frac{y_{i+1} - y_i}{d} - \frac{d^2}{2!\,d} y_i^{(2)} + \ldots;$$

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

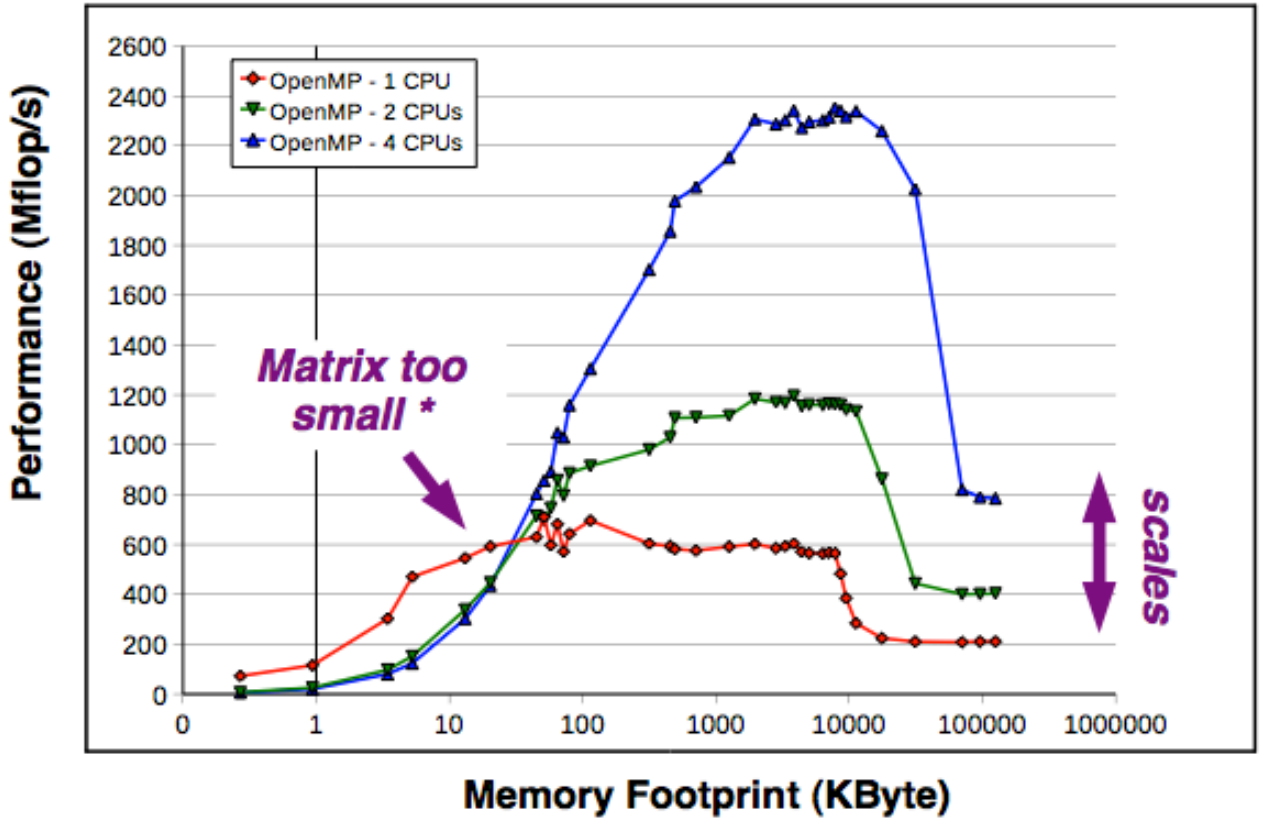
EJEMPLO DE PARALELISMO FUNCIONAL

```
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.



RENDIMIENTO DE OPENMP



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

Tomado de An Overview of OpenMP – SUN Microsystems

RESUMIENDO OPENMP

- OpenMP es una especificación que permite crear código concurrente desde un código secuencial, relativamente fácil.
- OpenMP permite la paralelización aprovechandose de los loops (ciclos) con fork-join:
- ```
  #pragma omp parallel
  ```
- ```
  #pragma omp parallel for
  ```
- ```
  #pragma omp parallel private ( i, x )
  ```
- ```
  #pragma omp atomic
  ```
- ```
  #pragma omp critical
  ```
- ```
  #pragma omp for reduction(+ : sum)
  ```
- OpenMP permite fácilmente identificar los hilos

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.

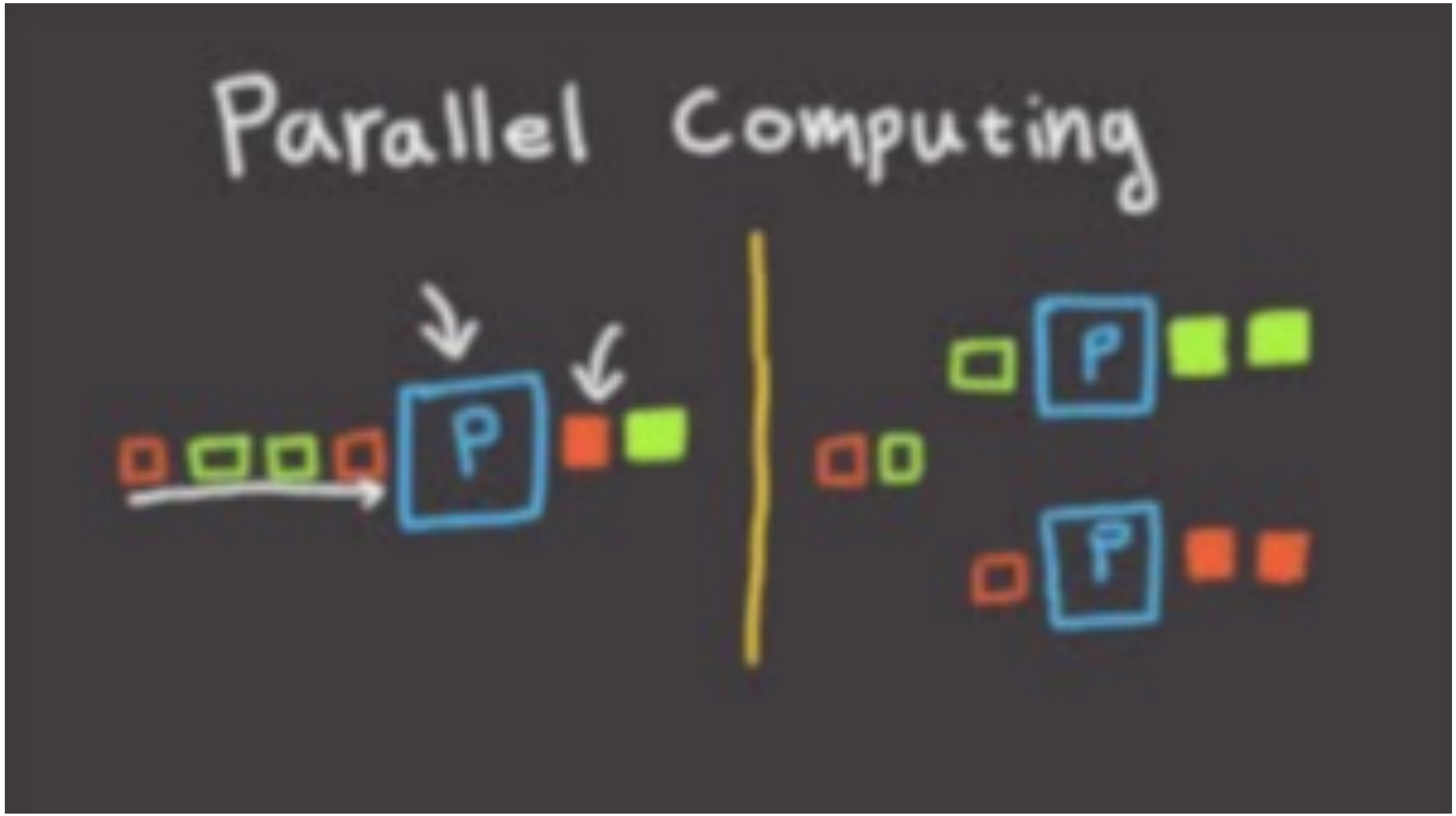
LECTURAS RECOMENDADAS

- 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/

TRABAJO PRÁCTICO

- En Clase:
  - Siga el tutorial : https://computing.llnl.gov/tutorials/openMP/ usando su cuenta en GUANE-1, corriendo todos los ejercicios.

- En Casa
  - Lea los articulos recomendados en las diapositivas anteriores (no olvidar los de las diapostivas 4, 5 y 6).



No worry, it's not clear (not yet)…