OpenMP

Sistemas Operativos II
Ingeniería en Computación

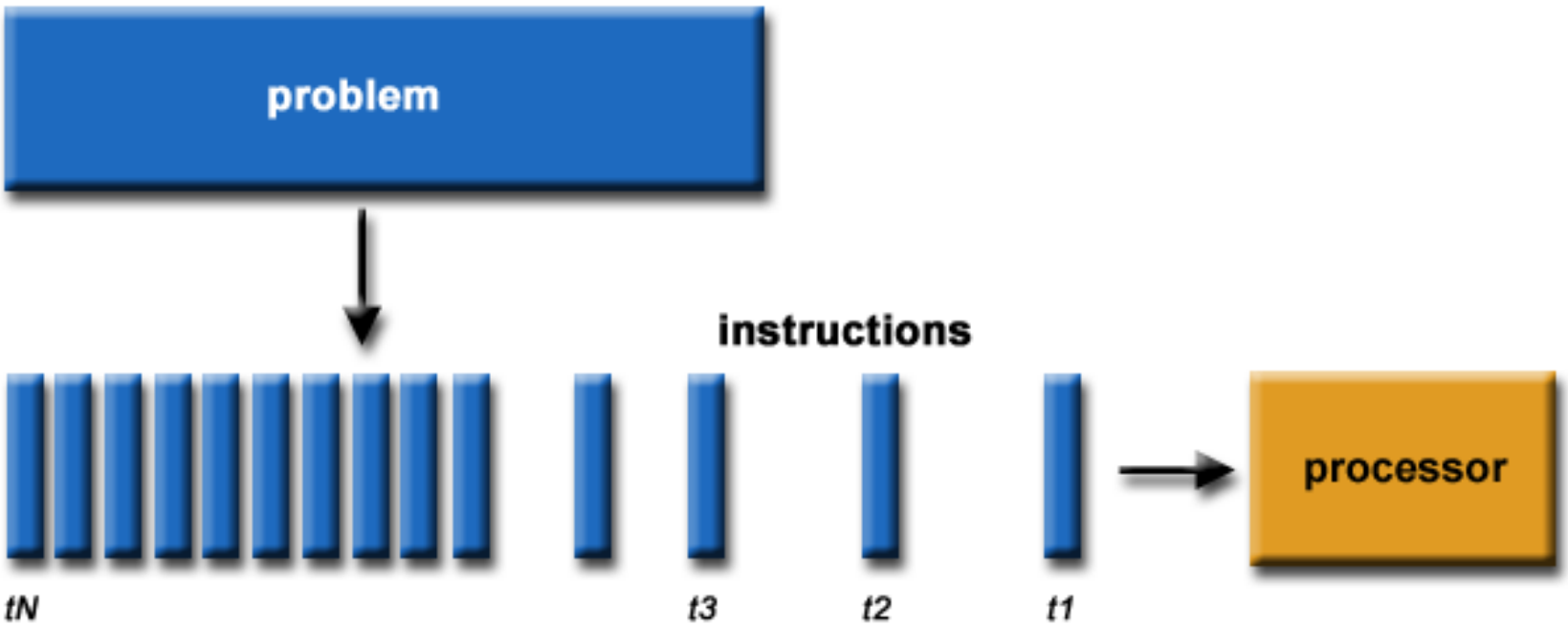
Universidad Nacional de Córdoba

Abril de 2016

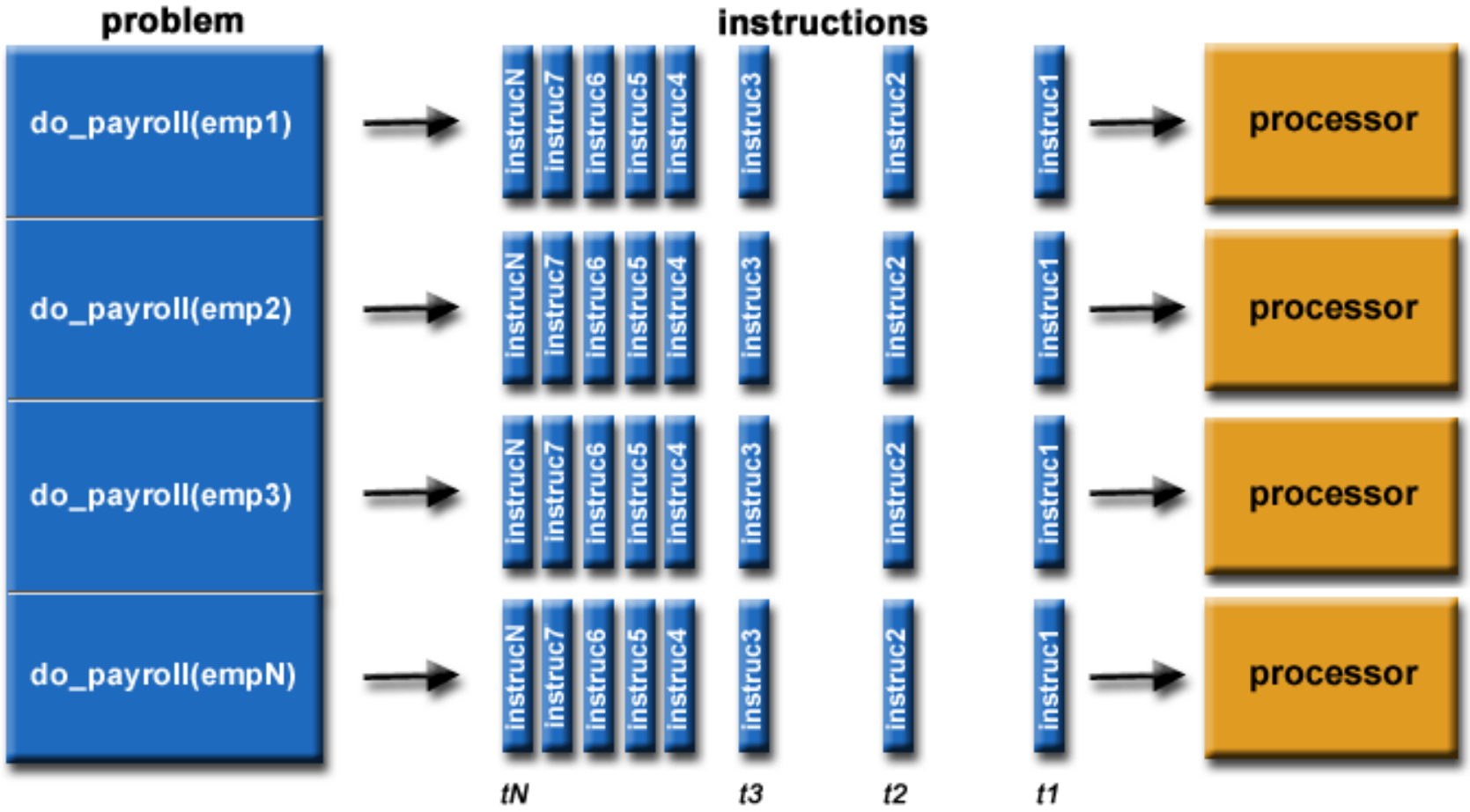
Temario

- Introducción
- Sintaxis
- Sincronización
- Variables compartidas
- Scheduling

Procesamiento lineal



Procesamiento Paralelo



Programación Paralela

- Tipo de cómputo donde los cálculos se hacen de forma simultánea[1]
- *"divide y vencerás"*
- **Paralela != Distribuida**

[1] Gottlieb, Allan; Almasi, George S. (1989). Highly parallel computing. Redwood City, Calif.: Benjamin/Cummings. ISBN 0-8053-0177-1

Distintos niveles de paralelismo

Paralelismo a:

- nivel de bit
- nivel de instrucción
- nivel de datos
- nivel de tareas

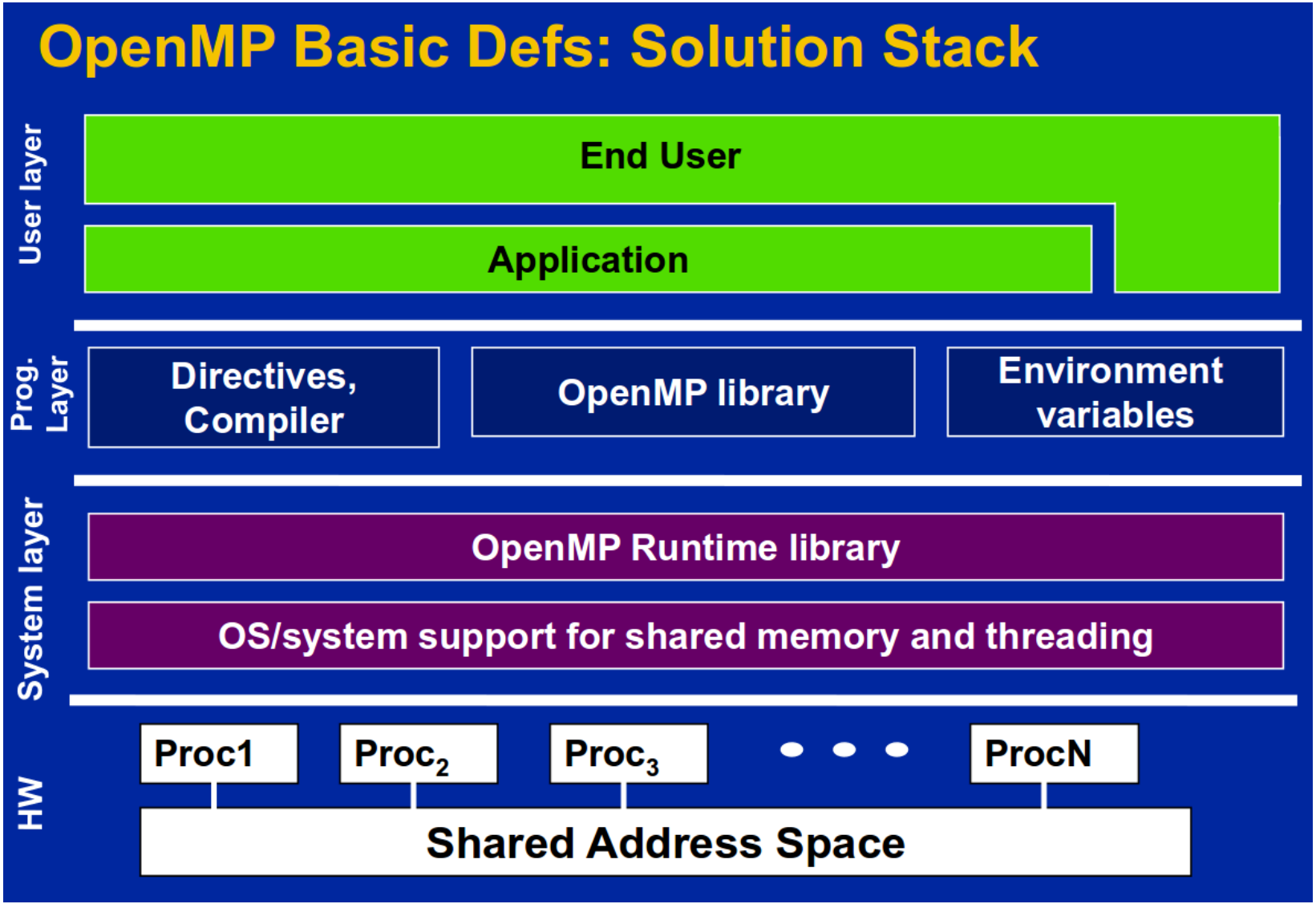
¿Qué es OpenMP?

- API multiplataforma para creación de programas multi-Threaded y de memoria compartida (paralelismo de datos, tareas).
- OpenMP Architecture Review Board (Intel, AMD, ARM, HP, Nvidia, etc.), cuyo objetivo es definir el estándar, administrar, promover y brindar soporte.
- Soporte de C, C++ y Fortran.

La API de OpenMP

- Conjunto de **directivas de compilador, rutinas de librería y variables globales**.
- Simplifica la escritura de programas multi-threaded.
- Estandariza prácticas que venían utilizándose.
- Memoria compartida permite compartir resultados entre los distintos procesos, no mensajes.

Stack


OpenMP Basic Defs: Solution Stack
User layer
End User
Application
Prog. Layer
Directives, Compiler
OpenMP library
Environment variables
System layer
OpenMP Runtime library
OS/system support for shared memory and threading
HW
Proc1
Proc2
Proc3
ProcN
Shared Address Space

Sintaxis

- En su mayoría, directivas de compilador
  - $\#pragmaomp < construct > [< arg > [, < arg >]]$
- Debe incluirse omp.h
  - $\#include < omp.h >$
- La mayoría de los constructs aplican a un bloque inmediatamente posterior

Ejercicio

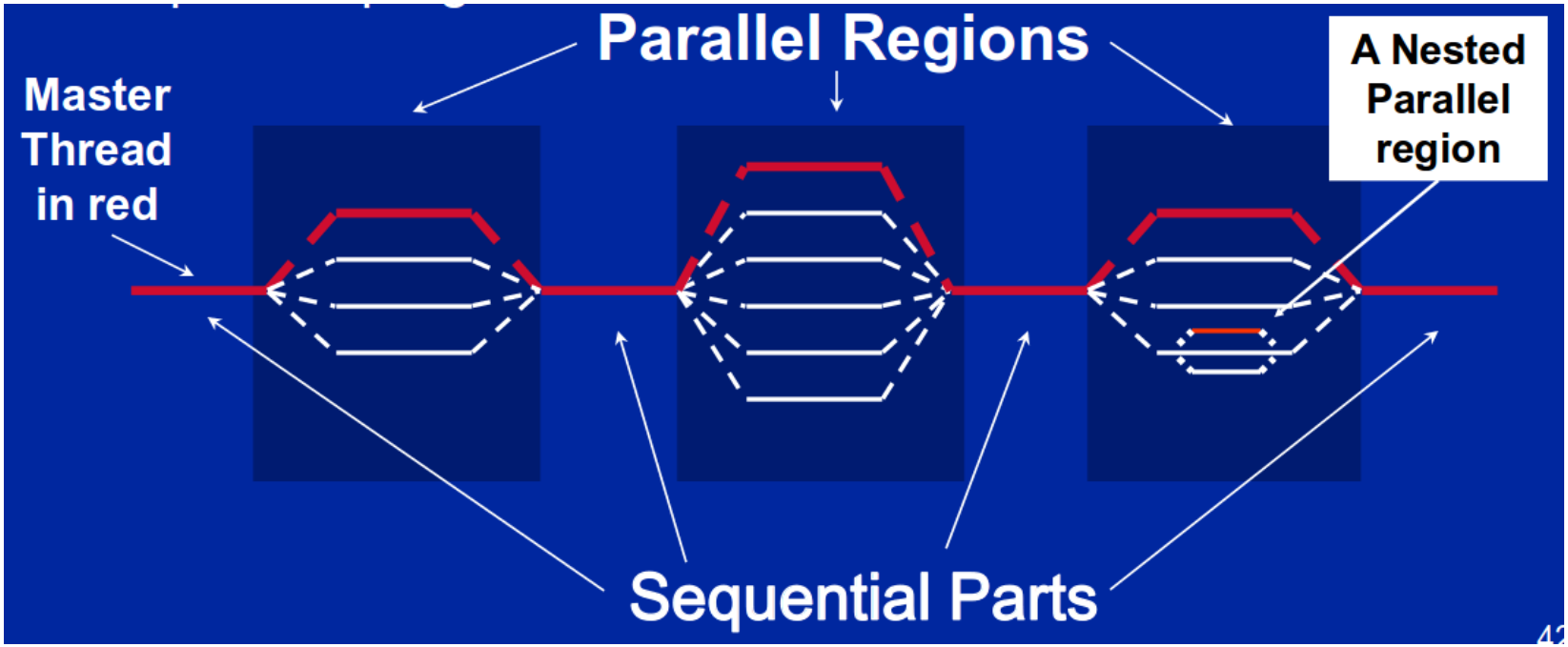
- Escribir un “Hello, World!” común.
- Agregar construcciones básicas de OpenMP
  - $\#include\ <omp.h>$
  - $\#pragma\ omp\ parallel$
  - Incluir la flag de gcc $-fopenmp$

¿Cómo interactúan los threads?

- Modelo multi-threading con memoria compartida (así es como se comunican).
- Se generan race conditions cuando la memoria se comparte inintencionalmente.
- Debe usarse sincronización para impedir las race conditions, aunque es costoso.
- Sólo debe recurrirse a la sincronización cuando se hayan agotado las posibilidades de reescribir el acceso a memoria.

Modelo de programación

- Hay un master thread que invoca a los demás threads.
- Los threads se invocan con el construct parallel.



Ejercicio

- Crear un programa que invoque un número arbitrario de threads.
- Cada thread debe escribir en stdout su thread id
  - $omp\_get\_thread\_num()$
- La cantidad de threads puede modificarse con
  - Argumento $num\_threads(N)$ luego del construct
  - $omp\_set\_num\_threads(N)$ llamada a función de librería

Sincronización

- Alto nivel
  - critical
  - atomic
  - barrier
  - ordered
- Bajo nivel
  - flush
  - locks

Sync: Critical

- Permite indicar una instrucción para ser ejecutada en exclusión mutua (mutex)

```
#pragma omp parallel
{
...
non_critical_func1(...);
...
#pragma omp critical
critical_func(...);
}
```

Sync: atomic

- Provee de mutex pero sólo para la actualización de una posición de memoria

```
...
#pragma omp atomic
sum+ = calc(...);
...
```

Sync: atomic

- Crear un programa en el que cada thread incremente una misma variable compartida (inicializada a 0).
- Comparar resultados con y sin uso de **critical/atomic**.

Parallel for

- Una forma de ejecutar el mismo código sobre diferentes datos (i.e. SIMD) sin redundancia de ejecución es separar la ejecución de un for-loop en chunks para cada thread. Es decir, se distribuye la carga.
- La variable de iteración se mantiene privada para cada thread, con lo cual se pueden tomar decisiones dentro del loop en base a la misma.
- Al final del loop hay sincronización implícita (i.e. barrier) para todos los threads.

Ejemplo

Sequential code

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

OpenMP parallel region

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

OpenMP parallel region and a worksharing for construct

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

Trabajando con loops

- Encontrar loops de cómputo intensivo.
- Hacer que cada iteración sea independiente del resto, para poder ejecutar cada una en un orden aleatorio.
- Incluir las directivas de OpenMP y probar.

Ejemplo

```
int i, j, A[MAX];
j = 5;
for(i = 0; i < MAX; ++i){
j+= 2;
A[i] = big(j);
}
```

```
int i, A[MAX];
#pragma omp parallel for
for(i = 0; i < MAX; ++i){
j+= 5 + (2 * i);
A[i] = big(j);
}
```

Reduction

- Cuando se tiene una variable cuyo valor se modifica en cada iteración, hay una dependencia de iteraciones que no puede ser eliminada trivialmente.

$$
\begin{aligned}
&double\ ave = 0{,}0,\ A[MAX];\ inti;\\
&for(i = 0; i < MAX; i++)\{\\
&\qquad ave+ = A[i];\\
&\}\\
&ave = ave/MAX;
\end{aligned}
$$

Reduction (cont...)

- Para ello, OpenMP (al igual que muchos otros entornos de programación paralela) incluyen la posibilidad de hacer reducciones (i.e. reductions).

Ejemplo

$$
\begin{aligned}
&double\ sum = 0{,}0,\ \ A[MAX];\\
&inti;\\
&\#pragma\ omp\ parallel\ for\ reduction\ (+ : sum)\\
&for(i = 0; i < MAX; ++i)\{\\
&\qquad sum+ = A[i];\\
&\}\\
&sum/ = MAX;
\end{aligned}
$$

Ejercicio

- Cálculo de π mediante el método de Montecarlo.

Más sincronización

- barrier
  - Cada thread espera al resto antes de continuar
  - *#pragma omp barrier*
- nowait
  - Elimina las barrier implícitas (e.g. parallel for)
  - *#pragma omp for nowait*
- master
  - Señaliza código a ser ejecutado sólo por el master thread
  - *#pragma omp master*

Más sincronización

- single
  - Sólo un thread puede ejecutar el código subsiguiente.
  - Hay un barrier implícito al final del bloque (puede eliminarse con nowait).
  - *#pragma omp single*
  - *#pragma omp nowait*
- ordered
  - La región se ejecuta en forma secuencial, en el orden indicado por los thread ID's.
  - *#pragma omp ordered*

Variables de entorno

- Relativo al número de threads
  - *omp_set_num_threads*()
  - *omp_get_num_threads*()
  - *omp_get_thread_num*()
  - *omp_get_max_threads*()
- Para saber si estamos en una región paralela
  - *omp_in_parallel*()
- Número de procesadores disponibles
  - *omp_num_procs*()

Variables de entorno

- Configuración dinámica del número de threads
  - *omp_set_dynamic*()
  - *omp_get_dynamic*()
- y más...

Data Sharing

- Las variables declaradas en el programa pueden o no ser compartidas por los threads.
- Por defecto, la mayoría de las variables son compartidas.
- Excepciones a esta regla:
  - Variables declaradas en una función llamada desde una region paralela.
  - Variables automaticas dentro de un bloque paralelo.
- y más...

Data Sharing

- Para cambiar los atributos de las variables, se usan los siguientes constructs
  - *shared* (default, toma la variable como compartida)
  - *private* (crea una instancia de la variable para cada thread)
  - *firstprivate* (inicializa la variable privada al valor que tenía al ingresar a la región paralela)

Data Sharing

- *lastprivate* (escribe a la variable global el último valor seteado en la región paralela)
- *default* (shared — private — none) (configura el comportamiento por defecto)

Scheduling

- Afecta el modo en el que las iteraciones son asociadas a cada thread.
  - *schedule*(*static*[, *chunk*])
    - Se reparten iteraciones de longitud chunk a cada thread.
  - *schedule*(*dynamic*[, *chunk*])
    - Cada thread toma chunk iteraciones de una queue.

Bibliografía

1. http://openmp.org
2. http://openmp.org/mp-documents/OpenMP_Examples_4.0.1.pdf