Computación Paralela

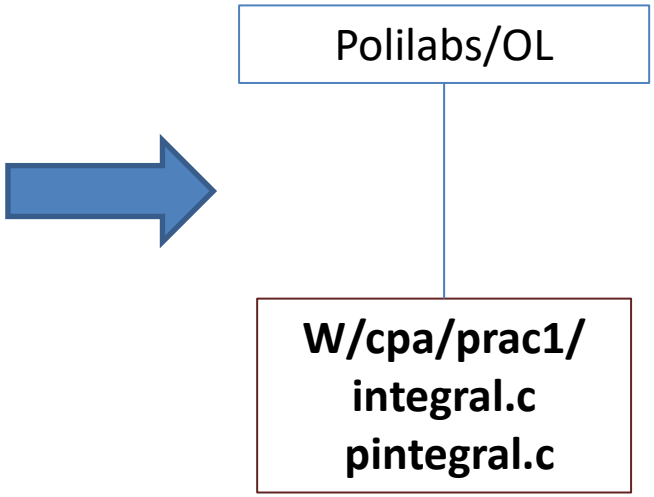
1ª Sesión de la práctica 1

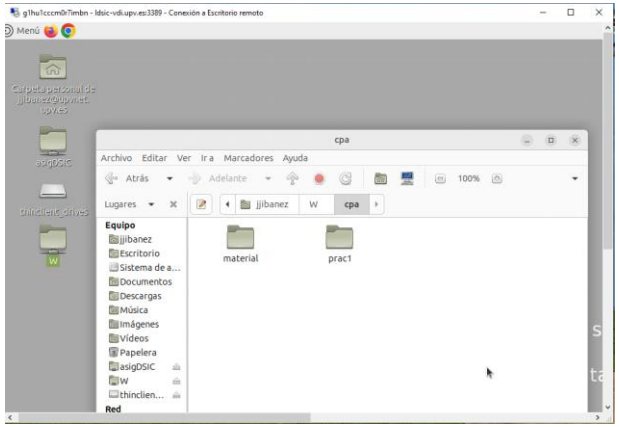
Objetivos

- Conocer el cluster **kahan** de prácticas
- Paralelizar mediante **OpenMp** los códigos secuenciales de dos funciones que calculan integrales definidas
- Mostrar el número de hilos que se ejecutan en la región paralela
- Medir tiempos en **OpenMP**
- Manejar las colas de **Kahan**
- **Nota:** Los boletines de prácticas y los ficheros de los códigos secuenciales de todas las prácticas se encuentran en **Tareas** de **Poliformat** de la asignatura

Primeros pasos

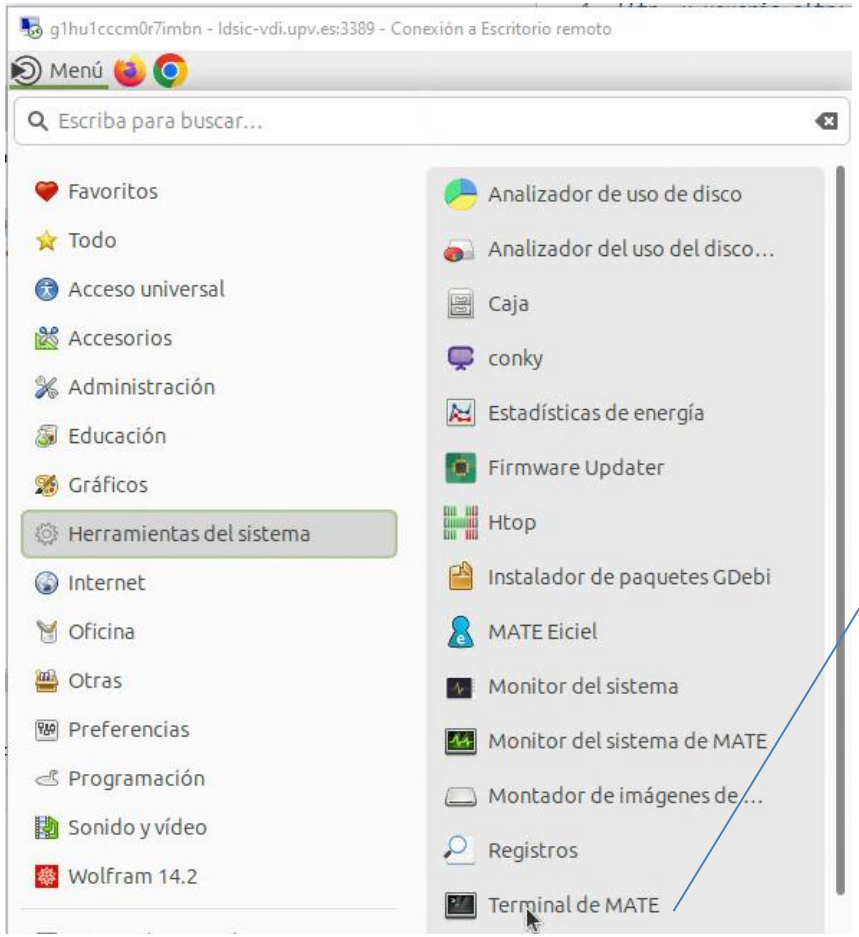
- Crea desde **Polilabs** o desde el ordenador laboratorio (**OL**) una carpeta en la unidad **W** para las prácticas de CPA, por ejemplo: **W/cpa/prac1**
- **Nota**: no debes dejar ningún espacio en blanco en los nombres de los directorios
- Copia en esa carpeta los ficheros de la primera práctica situados en **Tareas** de **Poliformat**, descomprimiendo en ella material-p1.tgz
- Copia el fichero **integral.c** en la carpeta de **W** con nombre **pintegral.c**.
- El fichero creado **pintegral.c** será el que tendréis que paralelizar

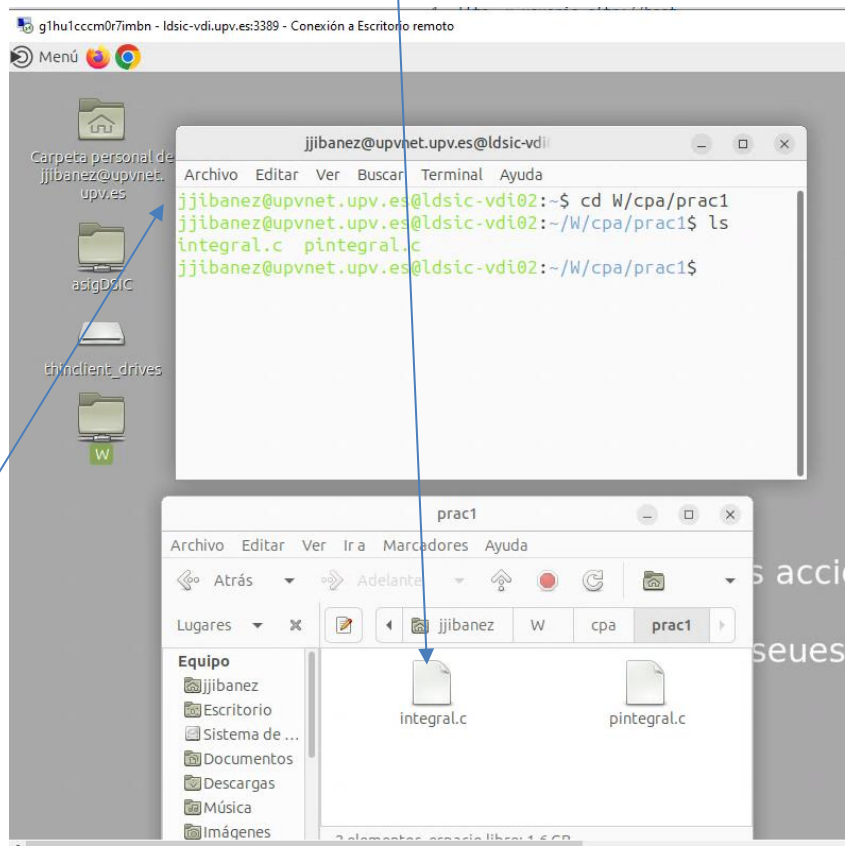




Primeros pasos

- Una vez has copiado el material, abre un terminal, por ejemplo, terminal **Mate**, puedes compilar y ejecutar los códigos de la práctica
- La edición del código de **pintegral.c** la puedes realizar desde el propio entorno gráfico, haciendo doble clic sobre el icono correspondiente



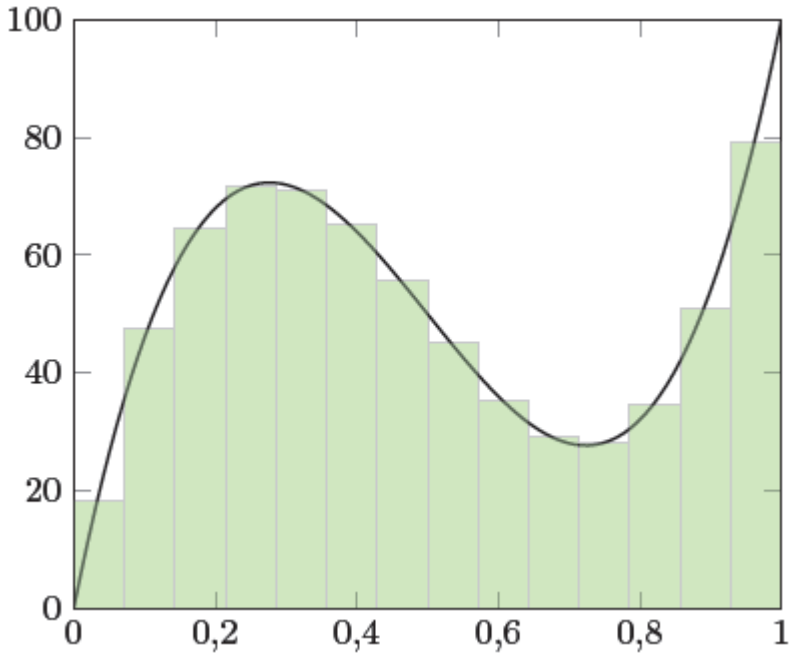


Código para mostrar el número de hilos

- Edita el programa **pintegral.c** desde **Polilabs** o desde el ordenador del laboratorio para que muestre el número de hilos con los que se ejecuta el programa paralelo
  - Deberás crear una región paralela en el **main**, por ejemplo, después de las declaraciones de las variables, de manera que se muestre mediante un **printf** el número de hilos
  - Recuerda añadir #include <omp.h> al principio del fichero

```
……
#include <omp.h>
….
int main(int argc, char *argv[]){
….. /*Declaración de variables*/
#pragma omp parallel
  {
   int id=omp_get_thread_num();
   if (id==0)
      printf("Numero de hilos: %d\n",  omp_get_num_threads());
  }
…..
}
```

Código secuencial integración numérica



$$\int_a^b f(x)dx \approx h\sum_{i=0}^{n-1} f(x_i)$$

$$x_i = a + h(i + 0.5)$$

Notas:

- El valor de la integral se aproxima mediante la suma de las áreas de los rectángulos de base igual a h=(b-a)/n (n=nº subintervalos) y altura igual a las imágenes de los puntos medios de cada uno de los subintervalos
- Cuantos más subintervalos tenga la partición, mayor será la precisión alcanzada

Programa integrales.c (Tareas Poliformat)

```
int main(int argc, char *argv[]){
  double a, b, result;
  int n, variante;
  if (argc<2) {
    fprintf(stderr, "Numero de argumentos incorrecto\n");
    return 1;
  }
  if (argc>2) n=atoi(argv[2]);
  else n=1000;
  a=0;  b=1;
  variante=atoi(argv[1]);
  switch (variante) {
    case 1:
      result = calcula_integral1(a,b,n);
      break;
    case 2:
      result = calcula_integral2(a,b,n);
      break;
    default:
      fprintf(stderr, "Numero de variante incorrecto\n");
      return 1;
  }
  printf("Valor de la integral = %.12f\n", result);
  return 0;
}
```

$$\int_a^b f(x)dx \approx h\sum_{i=0}^{n-1} f(x_i)$$

```
/* Calculo de la integral de una funcion f. Variante 1 */
double calcula_integral1(double a, double b, int n)
{
   double h, s=0, result;
   int i;
   h=(b-a)/n;
   for (i=0; i<n; i++) {
      s+=f(a+h*(i+0.5));
   }
   result = h*s;
   return result;
}
```

$$s = \sum_{i=0}^{n-1} f\big(a + h(i + 0.5)\big)$$

```
/* Calculo de la integral de una funcion f. Variante 2 */
double calcula_integral2(double a, double b, int n)
{
   double x, h, s=0, result;
   int i;
   h=(b-a)/n;
   for (i=0; i<n; i++) {
      x=a;
      x+=h*(i+0.5);
      s+=f(x);
   }
   result = h*s;
   return result;
}
```

$$x = a + h(i + 0.5)$$

$$s = \sum_{i=0}^{n-1} f(x)$$

La primera parte de la práctica consistirá en paralelizar los bucles de las dos funciones

Paralelización de la primera función

- Recuerda que tendrás que analizar el ámbito de las variables que aparecen en el bucle paralelo **for** y que pueden ser modificadas(variables i y s)
- Deberás añadir por tanto las cláusulas necesarias en el bucle paralelo (private, reduction, ….) de la primera función, para que esas variables tengan el comportamiento deseado

```
double h, s=0, result;
 int i;
 h=(b-a)/n;
 for (i=0; i<n; i++) {
   s+=f(a+h*(i+0.5));
 }
```

$$s = \sum_{i=0}^{n-1} f\left(a + h(i + 0.5)\right)$$

```
double h, s=0, result;
int i;
h=(b-a)/n;
#pragma omp parallel for …
for (i=0; i<n; i++) {
     s+=f(a+h*(i+0.5));
}
```

Completar

Paralelización de la segunda función

- De nuevo, deberás analizar el ámbito de las variables que hay dentro del bucle **for** de la segunda función (i, x y s)

```
double h, s=0, result;
int i;
h=(b-a)/n;
for (i=0; i<n; i++) {
      x=a;
      x+=a+h*(i+0.5)
      s+=f(x);
}
```

```
double x, h, s=0, result;
int i;
h=(b-a)/n;
#pragma omp parallel for .....
for (i=0; i<n; i++) {
      x=a;
      x+=a+h*(i+0.5)
      s+=f(x);
}
```

Completar

¿Qué ocurre con x?

$$x = a + h(i + 0.5)$$

$$s = \sum_{i=0}^{n-1} f(x)$$

Código para visualizar tiempos de ejecución

Para ver los tiempos de ejecución, añade en el programa principal las siguientes líneas de códigos:

```
variante=atoi(argv[1]);
double t;
t=omp_get_wtime();
switch (variante) {
    case 1:
      result = calcula_integral1(a,b,n);
      break;
    case 2:
      result = calcula_integral2(a,b,n);
      break;
    default:
      fprintf(stderr, "Numero de variante incorrecto\n");
      return 1;
  }
  t=omp_get_wtime()-t;
  printf("El tiempo de ejecución ha sido de %f segundos\n", t);
```

Compilación y ejecución de los códigos paralelos

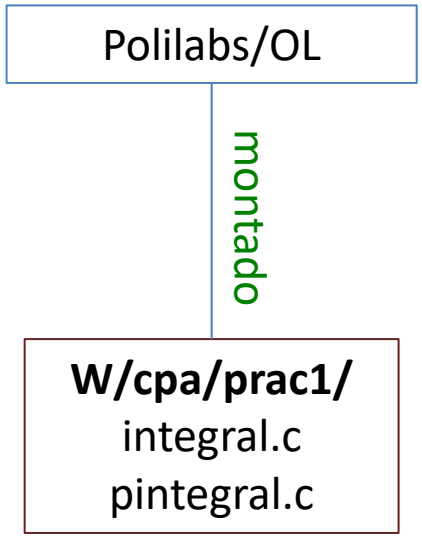
- Compila los códigos **integral.c** y **pintegral.c**:

```
gcc -Wall -o integral integral.c -lm
gcc -Wall -fopenmp -o pintegral pintegral.c -lm
```

- Ejecuta los códigos:

```
integral  1  10000
integral  2  10000
OMP_NUM_THREADS=4 ./pintegral  1  10000
OMP_NUM_THREADS=4 ./pintegral  2 10000
```

Deben dar lo mismo (puedes probar con un número diferente de hilos)



Ejecución en el Front-end de Kahan

- Conéctate al **Front-end** de kahan:

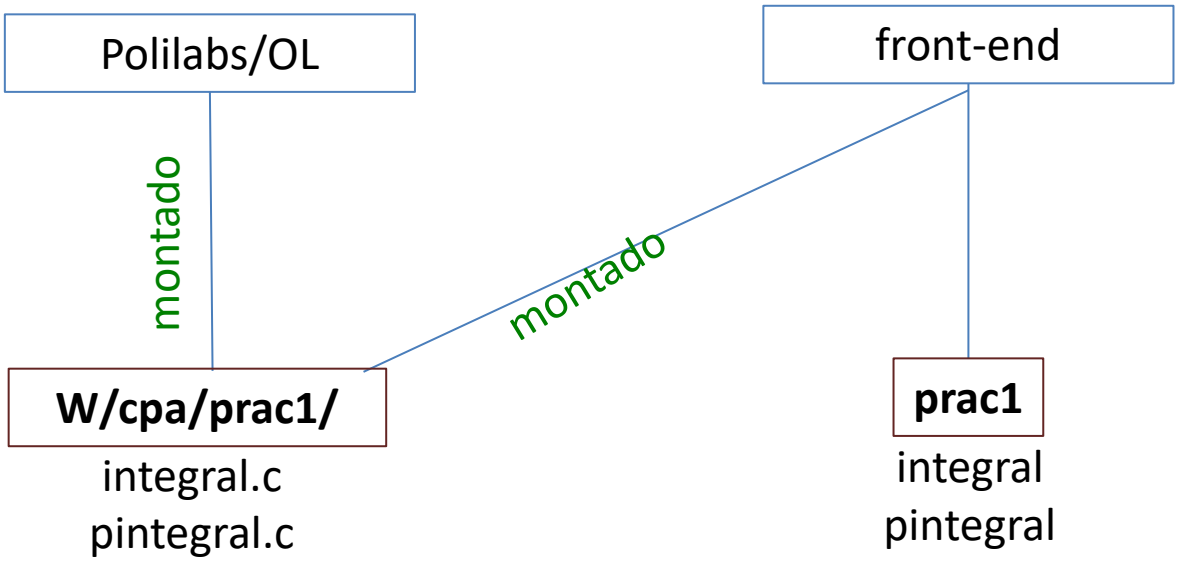
  ssh -l login@alumno.upv.es kahan.dsic.upv.es

- Crea desde el terminal del **front-end** la carpeta **prac1** y sitúate en ese directorio
- Compila de nuevo los programas, pero ahora en el **front-end** de Kahan:

  gcc -o integral ~/W/cpa/prac1/integral.c -lm

  gcc -fopenmp -o pintegral ~/W/cpa/prac1/pintegral.c -lm

- Ejecuta el programa en el f**ront-end**: ./pintegral 1 o ./pintegral 2



Ejecución en el Front-end de Kahan

- Otra posibilidad consiste en copiar los programas fuente integral. c y pintegral.c de W en la carpeta **prac1** del **front-end**:

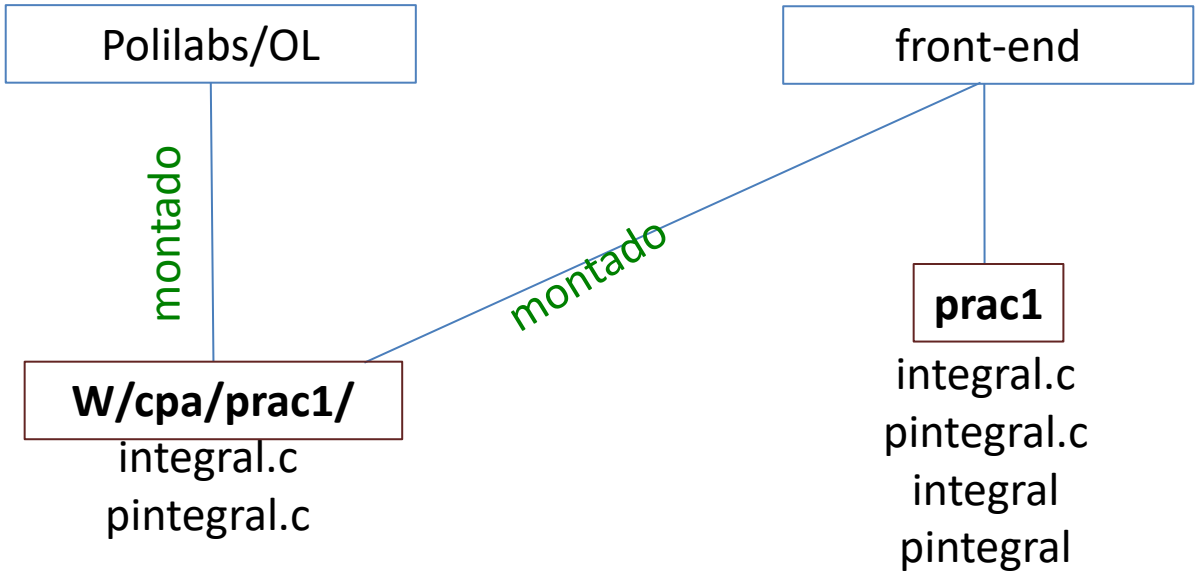
  cp ~/W/cpa/prac1/integral.c ~/prac1

  cp ~/W/cpa/prac1/pintegral.c ~/prac1

  y compilarlos desde la carpeta **prac1** del **front-end**:

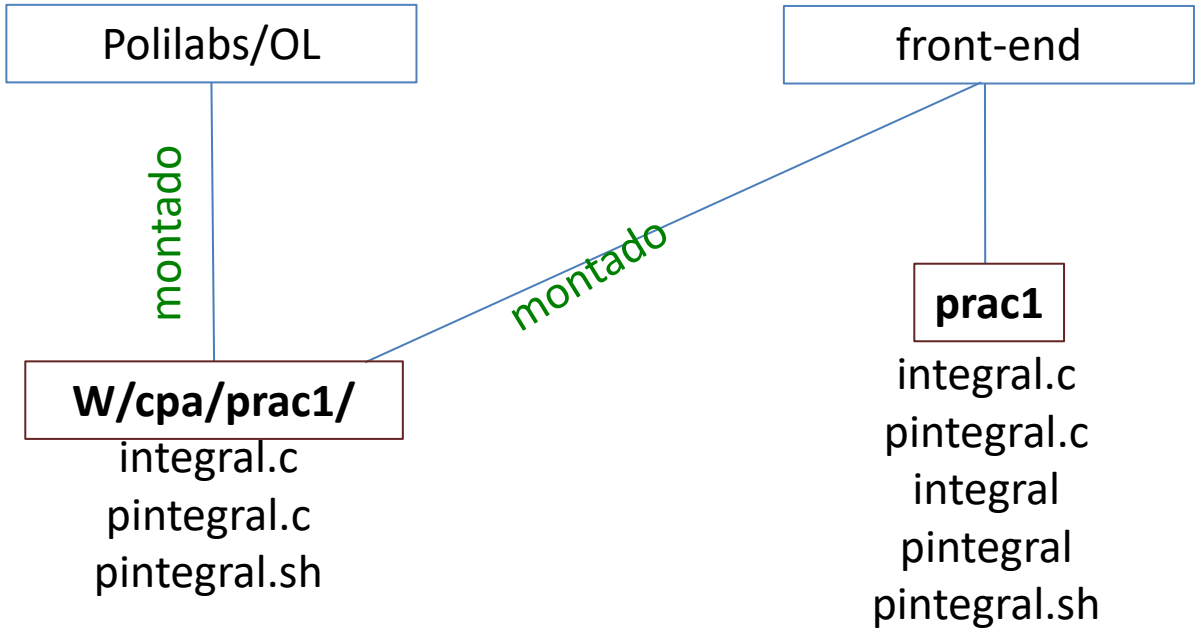
  gcc -Wall -fopenmp -o integral integral.c –lm

  gcc -Wall -fopenmp -o pintegral pintegral.c –lm



Lanzar trabajos en el cluster

- Para ejecutar un código en el cluster, debes usar el sistema de colas de colas **SLURM** de Kahan: deberás editar un script en la carpeta **prac1** de **W** y copiarlo al **front-end** de Kahan o directamente editarlo en el **front-end**



- Si has editado desde el ordenador del laboratorio o de la máquina remota el script pintegral.sh en W, lo debes copiar en el directorio que has creado en el Front-end de Kahan:

cp ~/W/cpa/prac1/pintegral.sh . (si está en el directorio prac1)

o

cp ~/W/cpa/prac1/pintegral.sh ~/prac1

Lanzar trabajos en el cluster

- Contenido del script pintegrales.sh:

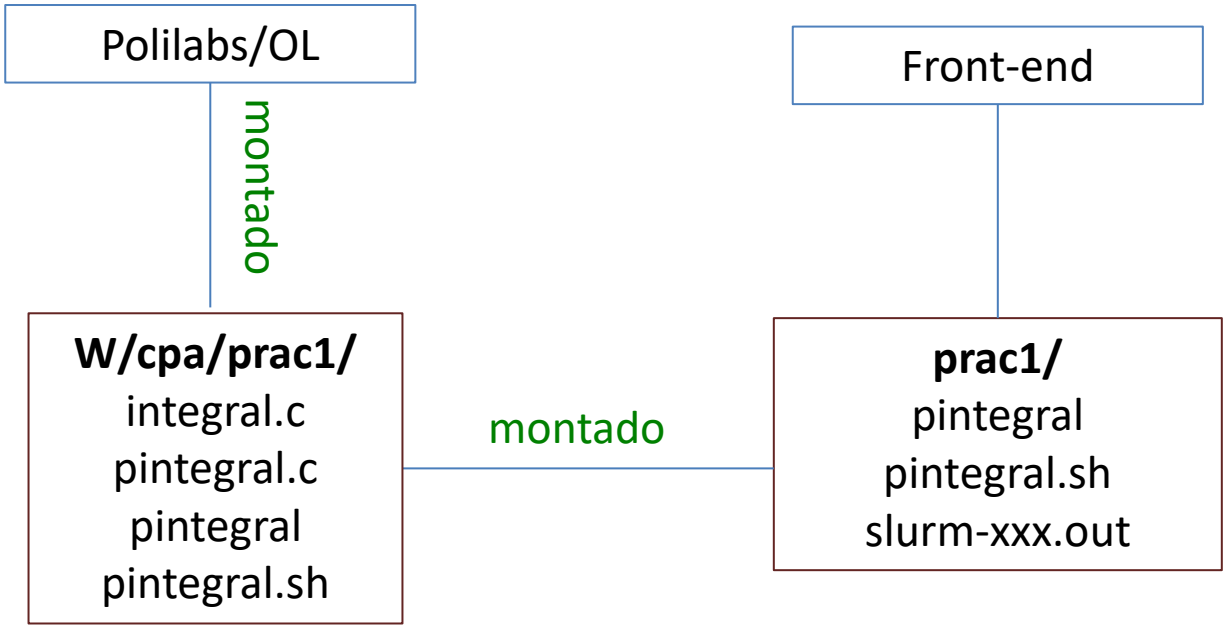
```
#!/bin/bash
#SBATCH --output=res_s1.txt
#SBATCH --nodes=1
#SBATCH --time=5:00
#SBATCH --partition=cpa
OMP_NUM_THREADS=1 ./pintegral 1 500000000000
OMP_NUM_THREADS=2 ./pintegral 1 500000000000
OMP_NUM_THREADS=4 ./pintegral 1 500000000000
OMP_NUM_THREADS=8 ./pintegral 1 500000000000
OMP_NUM_THREADS=16 ./pintegral 1 500000000000
OMP_NUM_THREADS=32 ./pintegral 1 500000000000
OMP_NUM_THREADS=64 ./pintegral 1 500000000000
```

1.3 Ejecución en el cluster

- Para lanzar el trabajo ejecuta el siguiente comando en el Front-end:

  sbatch pintegrales.sh

- Otros comandos interesantes para el manejo de colas:
  - **squeue**: permite ver el estado de la cola.
  - **scancel**: permite eliminar un trabajo de la cola perteneciente al usuario.
- Normalmente, el resultado de la ejecución lo tendrás en un fichero de nombre slurm-xxx.out, donde xxx corresponde al número que identifica al trabajo. Como has usado el comando #SBATCH --output=res_s1.txt, entonces los resultados los tendrás en el fichero res_s1.txt
- Usa el comando **cat** para ver el resultado.



Resultados

```
Numero de hilos: 1
El tiempo de ejecuci◆n ha sido de 32.608342 segundos
Valor de la integral = 1.000000000002
Numero de hilos: 2
El tiempo de ejecuci◆n ha sido de 17.702126 segundos
Valor de la integral = 1.000000000001
Numero de hilos: 4
El tiempo de ejecuci◆n ha sido de 10.247478 segundos
Valor de la integral = 1.000000000001
Numero de hilos: 8
El tiempo de ejecuci◆n ha sido de 5.202662 segundos
Valor de la integral = 1.000000000000
Numero de hilos: 16
El tiempo de ejecuci◆n ha sido de 3.121128 segundos
Valor de la integral = 1.000000000000
Numero de hilos: 32
El tiempo de ejecuci◆n ha sido de 1.682522 segundos
Valor de la integral = 1.000000000000
Numero de hilos: 64
El tiempo de ejecuci◆n ha sido de 1.105122 segundos
Valor de la integral = 1.000000000000
Numero de hilos: 128
El tiempo de ejecuci◆n ha sido de 1.109550 segundos
Valor de la integral = 1.000000000000
Numero de hilos: 256
El tiempo de ejecuci◆n ha sido de 1.034520 segundos
Valor de la integral = 1.000000000000
```

Muchas gracias
y
¡ a programar !